

# GRAPH-BASED LEARNING FROM LARGE IMAGE COLLECTIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Song Cao

May 2015

© 2015 Song Cao

ALL RIGHTS RESERVED



# GRAPH-BASED LEARNING FROM LARGE IMAGE COLLECTIONS

Song Cao, Ph.D.

Cornell University 2015

With the explosion of online images, it has been increasingly interesting for computer vision researchers to model large collections of images using machine learning techniques. Images, or observations by cameras in the real world, are natural units of information that are correlated to each other. For example, one of the ways that such correlation can be established is to check if two images observe the same part of the world (i.e. geometrically consistent). Hence, it is attractive to model images as well as their relationships with graphs. To achieve this goal, we need to first answer a few questions. First, how do we define such graphs and how do we acquire them? Second, how should we use such graphs to formulate learning such that the results are useful for computer vision tasks? Third, for large image sets, can we find ways to model the information in the set with a much smaller graph?

This thesis attempts to answer these questions with three corresponding chapters. In Chapter 2, we define the image graph as images (nodes) connected by an edge if and only if they are geometrically consistent, i.e. they have significant overlap. Chapter 2 will focus on how to acquire such graphs efficiently given a raw set of images. In short, our approach processes a set of images in an iterative manner, alternately performing pairwise feature matching and learning an improved similarity measure.

In Chapter 3, we formulate a learning problem making use of such image graph on a set of Internet tourist images for the task of location recog-

nition. In particular, starting from a graph based on visual connectivity, we propose a method for selecting a set of overlapping subgraphs and learning a local distance function for each subgraph using discriminative techniques. We demonstrate that our methods improve performance over standard bag-of-words methods on several existing location recognition datasets.

Finally, we propose a method for reducing the size of a Structure from Motion model using an image-point visibility graph in Chapter 4, and we show that this method produces small models that yield better recognition performance than previous model reduction techniques.

## **BIOGRAPHICAL SKETCH**

Song Cao finished his Ph.D. in the Computer Science in Cornell University in 2015. His Ph.D. research is on computer vision. Before coming to Cornell, he received his Bachelor of Engineering degree in Computer Software from Tsinghua University in China in 2008. He was originally born and raised in Hubei Province, China.

I dedicate this thesis to my parents.

## ACKNOWLEDGEMENTS

I want to thank my advisor Professor Noah Snavely for his great support, patience and guidance for me during my five and a half years of PhD life. Though the course of my work with Professor Snavely, he has always been available to offer helpful suggestions and constructive thoughts. I have learned a lot from him about research, computer vision and life. I'm deeply grateful for all he has done for me. I have also had the enormous pleasure to know and work with Professor Kavita Bala and fellow lab mates. I would also like to thank Professor Thorsten Joachims and Professor Peter Frazier, whose helpful input for my research is greatly appreciated.

I would also like to thank my friends during my PhD study, who have made my time at Cornell such a pleasant experience. Finally, I would like thank my parents, especially my mother, whose love and support for me mean the world to me.

## TABLE OF CONTENTS

Biographical Sketch . . . . .	iii
Dedication . . . . .	iv
Acknowledgements . . . . .	v
Table of Contents . . . . .	vi
List of Tables . . . . .	viii
List of Figures . . . . .	x
<b>1 Introduction</b>	<b>1</b>
<b>2 Learning to Match Large Scale Image Collections</b>	<b>5</b>
2.1 Related Work . . . . .	9
2.2 Our Approach . . . . .	11
2.2.1 Discriminative Learning of a Classifier for Image Pairs . .	11
2.2.2 Iterative Learning and Matching . . . . .	15
2.3 Experiments . . . . .	16
2.3.1 Performance of Discriminative Learning . . . . .	16
2.3.2 System Evaluation . . . . .	22
<b>3 Graph-Based Discriminative Learning for Location Recognition</b>	<b>26</b>
3.1 Related Work . . . . .	29
3.2 Graph-based Location Recognition . . . . .	31
3.3 Graph-based Discriminative Learning . . . . .	34
3.4 Diverse Shortlists . . . . .	45
3.4.1 Probabilistic Reranking . . . . .	45
3.4.2 Bag-of-Words Regularization . . . . .	50
3.5 Evaluation . . . . .	53
3.5.1 Datasets and Preprocessing . . . . .	53
3.5.2 Evaluation Methodology . . . . .	54
3.5.3 Runtime analysis . . . . .	57
3.5.4 Results . . . . .	58
<b>4 Minimal Scene Descriptions from Structure from Motion Models</b>	<b>64</b>
4.1 Related Work . . . . .	66
4.2 Computing Minimal Point Sets . . . . .	68
4.2.1 Maximizing expected coverage . . . . .	70
4.2.2 Appearance-aware initial point set selection . . . . .	72
4.2.3 Probabilistic $K$ -cover algorithm . . . . .	76
4.2.4 Full point set reduction algorithm . . . . .	80
4.3 Implementation . . . . .	81
4.4 Experiments . . . . .	83
<b>5 Conclusion</b>	<b>87</b>

<b>A</b>	<b>89</b>
A.1 Derivation for the iterative usage of the update factor . . . . .	89
<b>Bibliography</b>	<b>91</b>

## LIST OF TABLES

2.1	<b>mAP performance of models trained with various training data sizes.</b> The baselines are the mAP scores for rankings using <i>tf-idf</i> similarity and co-ocset similarity [10]. Columns marked with <b>200</b> , <b>1000</b> , <b>2000</b> , and <b>~100K</b> show the performance of models trained with corresponding number of examples. <b><i>N</i>+neg</b> corresponds to models trained with the same set of <i>N</i> examples combined with large amounts of negative examples from other datasets; <b><i>N</i>+mr</b> denotes models trained with the same set of <i>N</i> examples using our modified regularization. . . . .	19
2.2	<b>Improved mAP on all 7 datasets using modified regularization with limited training examples.</b> This table shows the performance of our learning method with the standard SVM regularization, with our modified regularization, and the baselines, <i>tf-idf</i> similarity and “co-ocset” similarity, in the task of ranking. “# Pairs” shows the number of training image pairs used for training corresponding models. The modified regularization performs better than either other method when small amounts of training data are available, though the performance gap between the two regularization approaches shrinks, as expected, when more training data is used. . . . .	21
3.1	<b>Summary of datasets used in our experiments, along with their computed neighborhoods.</b> Each row summarizes a dataset, showing the number of query images in the test set, the total number of images in the database, and the number of neighborhoods our algorithm selects in order to cover each image graph. The representative neighborhoods (clusters) are found using graphs whose edge weights are defined using the Jaccard index, and thresholded using a value of 0.01. The rightmost column shows the average cluster size in each dataset, along with the standard deviation in the cluster sizes. . . . .	54
3.2	<b>Recognition performance on Dubrovnik dataset.</b> The abbreviations for each method are defined in Section 3.5.2. . . . .	58
3.3	<b>Recognition performance on Rome, Aachen and Landmarks datasets.</b> The abbreviations for each method are defined in Section 3.5.2. . . . .	59
4.1	Summary of datasets used in our experiments. . . . .	83



4.2	<b>Registration performance on Dubrovnik, Aachen, and Landmarks datasets.</b> <b>KC</b> stands for the $K$ -cover algorithm, <b>KCD</b> stands for our appearance-aware point selection algorithm, and <b>KCP</b> stands for our full approach. Point sets of the same size are selected using the three algorithms, then used in the same registration algorithm [30] to evaluate the percentages of query images that are successfully registered to the database. Smaller $K$ values (in brackets) are used to initialize our KCP method. For each experiment, we show the number of points in the reduced model, the percentage of total points this represents, and the performance of the three methods. For comparison, we also show the performance of [30] using the full set of input points. . . . .	85
-----	---	----

## LIST OF FIGURES

1.1	<b>An example image graph.</b> Nodes in this graph are images, and edges connect visually overlapping images. . . . .	2
2.1	<b>Training an SVM classifier with positive and negative image pairs.</b> Although each image pair (images with borders of the same color) shown above contain images with many common visual words (shown as boxes with same colors each pair), some pairs are true matches (top right), while others are false matches (bottom left). Accordingly, some visual words are more discriminative (or confusing) than others. Our goal is to learn a weighting of different visual words to better predict matching and non-matching image pairs. This weighting is shown here as a max-margin separating hyperplane. The images shown above are from the <b>Trafalgar</b> dataset. . . . .	6
2.2	<b>Two example visual words with different discriminative power.</b> The three images on the left contain a common visual word (in green), which is highly weighted by our learned model. In contrast, the three images on the right also share a common visual word (in red), but do not match; this word is given low weight. . . . .	6
2.3	<b>Correct predictions of non-matching pairs.</b> Due to challenging differences in contrast, illumination and viewpoints, these two image pairs both failed the SIFT matching and verification process, despite exhibiting visual overlap (as well as common visual words, which are marked with boxes of the same color). In contrast, our model is able to correctly highly rank these images, as they happen to have very discriminative visual words (in red). Note that the common visual words may not always imply exact correspondence (e.g., because of repeating patterns). . . . .	14
2.4	<b>Histograms of matching vs. non-matching testing pairs.</b> From left to right: histograms of <i>tf-idf</i> similarity, co-ocset similarity [10] and our output values ( $\sim 100K$ training pairs) respectively, for matching and non-matching test pairs. Note the log scale on the <i>y</i> -axis. The test pairs consist of randomly chosen unseen image pairs from the <b>TateModern</b> dataset. . . . .	14
2.5	<b>Some example images from the 5 new datasets we used for evaluation.</b> Note that in these 5 datasets, there also exist many irrelevant images due to the noisiness inherent in Internet image collections. . . . .	17

2.6	<b>Per-round <i>match success rates</i> for five datasets.</b> The $x$ -axis is the round number, and the $y$ -axis is the percentage of image pairs tested so far found to be true matches. Since we use <i>tf-idf</i> similarity in the first round, the corresponding percentages are the same for that round. . . . .	23
2.7	<b>Number of successful matches as a function of CPU time.</b> This figure plots the numbers of true matches found as a function of CPU time spent, for both our approach and the baseline, for each dataset. After each round (shown by the data points above), the time for our approach is measured as the sum of image matching time, training time and re-ranking time, while that of baseline only includes image matching time. Our approach shows significant improvement in terms of efficiency in discovering true matching image pairs. . . . .	25
3.1	<b>A region of an example image graph with three neighborhoods defined by representative images A, B and C.</b> Nodes in this graph are images, and edges connect visually overlapping images. Our method uses the graph to find a set of representative neighborhoods (inside the dotted circles above) that cover the graph, and learns a local distance function for each neighborhood. These distance functions are used to connect a new query image (left) to the rest of the graph and hence recognize its location. Given a query image, we match to the graph using these learned neighborhood models, rather than considering database images individually. Each neighborhood has its own distinctive features, and our goal is to learn and use them to aid recognition. . . . .	27
3.2	<b>Image matching graph for the Dubrovnik dataset and selected exemplar nodes.</b> This graph contains 6,844 images (white dots); the large, red nodes denote representative exemplar images selected by our covering algorithm (478 images in total). Although the set of representative images is much smaller than the entire collection, their neighborhoods cover the matching graph. For each neighborhood, we learn a classifier for determining whether a new image belongs to that neighborhood. (Note that the layout of the graph is based on its structure, and not any underlying image locations.) . . . . .	38

3.3	<b>Two example neighborhoods produced by our covering algorithm.</b> Each row shows an example neighborhood. The leftmost image in each row is the exemplar (center) image for each neighborhood. The remaining images are sample members of that neighborhood, sorted in decreasing order by their edge weights (Jaccard index) with the center image. We have found that this way of representing locations—as collections of overlapping image neighborhoods—is a natural one, allowing us to use discriminative learning to learn the most representative features to best identify the location of a query image, adapting to each neighborhood. . . . .	39
3.4	<b>Visualization of learned SVM weights on two example neighborhoods.</b> Here, we show, for two neighborhoods, several example images from each neighborhood. The SIFT features shown in each image are color-coded to visualize the weights for their corresponding visual words, as learned through our SVM training process for each neighborhood. The closer to red the color, the higher the corresponding weight is; the closer to blue, the lower the weight is. Most of the features (typically > 99%) are assigned zero weights (and are not shown above), due to the $L_1$ regularization used for training our SVMs and due to the large vocabulary size (1M in our experiments, see Section 4.4). The features shown here are assigned high positive or negative weights. Note that most features have positive weights, whereas negative weights provides better separation between different neighborhoods. For example, in the top-right image of the neighborhood on the left, negative weights on the other side of the building belong to another neighborhood while features on the front side get highly weighted. In addition, note how repetitive features get mid-range positive weights while discriminative and distinctive features get highest weights. . . . .	42
3.5	<b>Two example query images and their top 5 ranked results using our learned similarities and raw tf-idf BoW retrieval.</b> For each result, a green border indicates a correct match, and a red border indicates an incorrect match. The two example query images on the left are difficult for BoW retrieval techniques, due to drastically different lighting conditions (query image 1) and confusing features (rooftops in query image 2). However, with our discriminatively learned similarity functions, correctly matching images are ranked higher than with the baseline method. . . . .	44

3.6	<b>An example query image and the top 5 ranking results using our method with and without probabilistic ranking.</b> Green borders indicate correct matches, and red borders incorrect ones. Without probabilistic ranking, our algorithm generates a top 5 set of results that are similar, and all incorrect. With probabilistic reranking, more diversity is encouraged in the top ranking results, leading to several correct images among the top 5 results.	50
3.7	<b>The overall procedure for our BoW-based regularization of the shortlists.</b> In each round of verification for the query image, we select the best scoring image (alternating between averaged probability and BoW similarity) in the <b>Final Ranking</b> step. If this selected image doesn't match the query, we update the average probabilities for the remaining images using Eq. (3.5).	52
3.8	<b>Top-k accuracy on the Dubrovnik dataset (generic vocabulary).</b>	61
4.1	<b>Distributions of descriptor distances.</b> The curve in red shows the distribution of descriptor distances between image feature descriptors and their associated point descriptor centroids. The curve in blue shows the distribution of distances between a point descriptor centroid and its nearest neighbor in the full database. Note the significant overlap between these distributions. These distributions are generated from the Dubrovnik dataset by considering all database points and their associated image feature descriptors. (Figure best viewed in color.)	73
4.2	<b>An illustration of appearance-aware point selection.</b> The image above shows points in feature descriptor space (reduced to 2D for visualization purposes). Blue triangles represent point descriptors that are already selected by our algorithm, while red circles represent descriptors of candidate points to select next. Suppose that candidate points A and B cover exactly the same number of images in $\mathcal{I} \setminus C$ , and thus would lead to equal gains in the $K$ -cover algorithm. However, since the minimum distance of A to the selected point set $d_{\min}(A)$ is larger than that of B $d_{\min}(B)$ , A is likely to result in fewer mismatches. during feature matching if selected. Hence A is preferred by our appearance-aware point selection algorithm.	74

- 4.3 **Distributions of distances with and without appearance-aware selection.** This plot illustrates the “density” of two sets of point descriptors, by showing the distribution of distances between nearest neighbors in SIFT space between points within each set. The blue plot is for the set selected with the basic  $K$ -cover algorithm (KC), while the green plot is for the set selected with our appearance-aware selection algorithm (KCD) using the threshold  $d = 180$ . Both point sets contain 311,343 points selected from the Landmarks dataset [30]. Note that the KCD algorithm pushes points further away from one another on average. . . . . 76
- 4.4 **Binomial distributions for images covered by different numbers of selected points.** The plots above show probability distributions for number of visible points,  $v_{i,\mathcal{P}'}$ , for three different images (solid lines). The images corresponding to the green, red solid, and blue curves are covered by 15, 20, and 35 points, respectively; in this example the probability of a positive point observation is set to  $p = 0.6$ . The legend shows the probability mass on the right side of the line  $K = 12$  for each image, i.e.  $\Pr(v_{i,\mathcal{P}'} \geq K)$ , the probability that each image sees at least  $K$  points in  $\mathcal{P}'$ . As more points are added, the distributions will shift from left to right (green  $\rightarrow$  red  $\rightarrow$  blue). The dotted red distribution is the red distribution after adding a single new point that is visible in this image; the corresponding probability value  $\Pr(v_{i,\mathcal{P}'} \geq K)$  has increased from 0.488 to 0.596 (a gain of 0.108). The objective of our probabilistic point selection algorithm is to select points that maximize the increase in expected gain  $\Pr(v_{i,\mathcal{P}'} \geq K)$  for all uncovered images. . . . . 78

## CHAPTER 1

### INTRODUCTION

A key problem in recent Web-scale vision systems is to take a large, unstructured image collection (e.g., a large set of Internet photos) and discover its visual connectivity structure, i.e., determine which images overlap which other images, in the form of an *image graph*, and find feature correspondence between matching images. Finding this structure often involves testing many pairs of images, by matching SIFT features [31] and performing geometric verification. For example, 3D reconstruction methods for large-scale Internet photos—such as all photos of Rome—require finding feature correspondence by matching many pairs of images [1, 15], and other applications, such as summarizing photo collections [50] and unsupervised discovery of objects [41] require similar connectivity information. Malisiewicz et al. formulates and improves object recognition using a graph structure “Visual Memex” [32]. For location recognition, Torii et al. propose an image graph regression technique which matches images to the graph [53]. Li et al. also make use of image-point visibility graph in direct matching between 2D and 3D points [30].

A commonly used image graph definition is that images are defined as nodes, and only geometrically consistent image pairs are connected with edges (e.g. Figure 1.1). As will be shown in Chapter 3, such a graph can be used to formulate discriminative learning to improve location recognition performance significantly.

However, the brute force approach to compute such a graph is a computationally expensive process, because the graph of matching image pairs is unknown in advance, and so methods for quickly and accurately predicting which

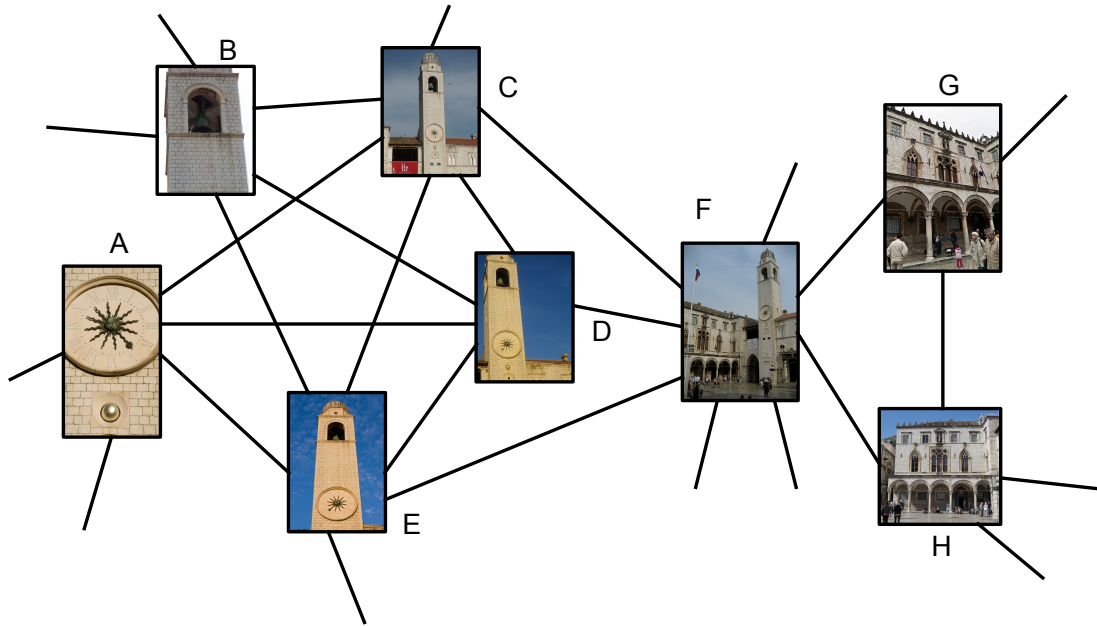


Figure 1.1: **An example image graph.** Nodes in this graph are images, and edges connect visually overlapping images.

of the  $\mathcal{O}(n^2)$  pairs of images match are critical.

Image comparison methods such as bag-of-words models [51] or global features are often used to predict similar pairs, but can be very noisy. In Chapter 2, we propose a new image matching method that uses discriminative learning techniques—applied to training data gathered automatically during the image matching process—to incrementally compute a better similarity measure for predicting whether two images in a given collection overlap. By using such a learned similarity measure, our algorithm can select image pairs that are more likely to match for performing further feature matching and geometric verification, improving the overall efficiency of the matching process.

Our approach processes a set of images in an iterative manner, alternately performing pairwise feature matching and learning an improved similarity



measure. Our experiments show that our learned measures can significantly improve match prediction over the standard *tf-idf*-weighted similarity and more recent unsupervised techniques even with small amounts of training data, and can improve the overall speed of the image matching process by more than a factor of two.

One application of such a computed image graph is to use it to improve location recognition performance, i.e. recognizing the location of a query image by matching it to an image database. This is an important problem in computer vision, and one for which the *representation* of the database is a key issue. In Chapter 3, we explore new ways for exploiting the structure of an image database by representing it as a graph, which can be computed efficiently using the techniques from Chapter 2, and show how the rich information embedded in such a graph can improve bag-of-words-based location recognition methods. In particular, starting from a graph based on visual connectivity, we propose a method for selecting a set of overlapping subgraphs and learning a local distance function for each subgraph using discriminative techniques. For a query image, each database image is ranked according to these local distance functions in order to place the image in the right part of the graph. In addition, we propose a probabilistic method for increasing the diversity of these ranked database images, again based on the structure of the image graph. We show in our experiments that this approach can improve the image-retrieval based location recognition system significantly.

Finally, a key challenge for many real-world applications is scalability. Structure from motion models can grow to a significant size given only a few thousand images, which begs the question: How much data do we need to describe a

location? In Chapter 4 we explore this question in the context of 3D scene reconstructions created from running structure from motion on large Internet photo collections, where reconstructions can contain many millions of 3D points. We consider several methods for computing much more compact representations of such reconstructions for the task of location recognition, with the goal of maintaining good performance with very small models. In particular, we introduce a new method for computing compact models that takes into account both image-point relationships and feature distinctiveness. The image-point relationships are represented as an image-point visibility graph, similar to [30]. Although the graph definition is different from the previous two chapters, the idea of analyzing and learning from graphs is universally useful in these cases.

## CHAPTER 2

### LEARNING TO MATCH LARGE SCALE IMAGE COLLECTIONS

As mentioned in Chapter 1, the computational cost for feature matching and geometric verification, which are essential in computing the image graph, can be quite high, especially if more than a small fraction of the total  $O(n^2)$  possible image pairs in a set of  $n$  images are matched. However, many large image collections exhibit sparse visual connectivity—only a fraction of possible image pairs overlap. The question is then: how can we compute a good approximation of the image connectivity graph, as efficiently as possible? We present a method that *learns* a good measure for comparing images during such an image matching process, improving this measure as it discovers the structure of the image graph.

To avoid exhaustive feature matching on all  $O(n^2)$  image pairs, recent work has used fast, whole-image similarity measures, such as bag-of-words (BoW) [51, 1, 15, 41] or GIST features [27], to predict a smaller set of candidate image pairs on which to perform detailed matching. BoW methods in particular, often used in image retrieval [38], have had increasing success for this image matching problem. However, BoW similarities are quite noisy, due to quantization error and imperfect feature detections. As a result, when used to predict image pairs for matching, many cycles are wasted matching features between non-overlapping images, making the matching process unnecessarily time-consuming.

In this chapter, we explore a new, iterative approach that learns to predict which pairs of images in an input dataset match, and which do not, using discriminative learning of BoW models. Our method adapts over time in the pro-

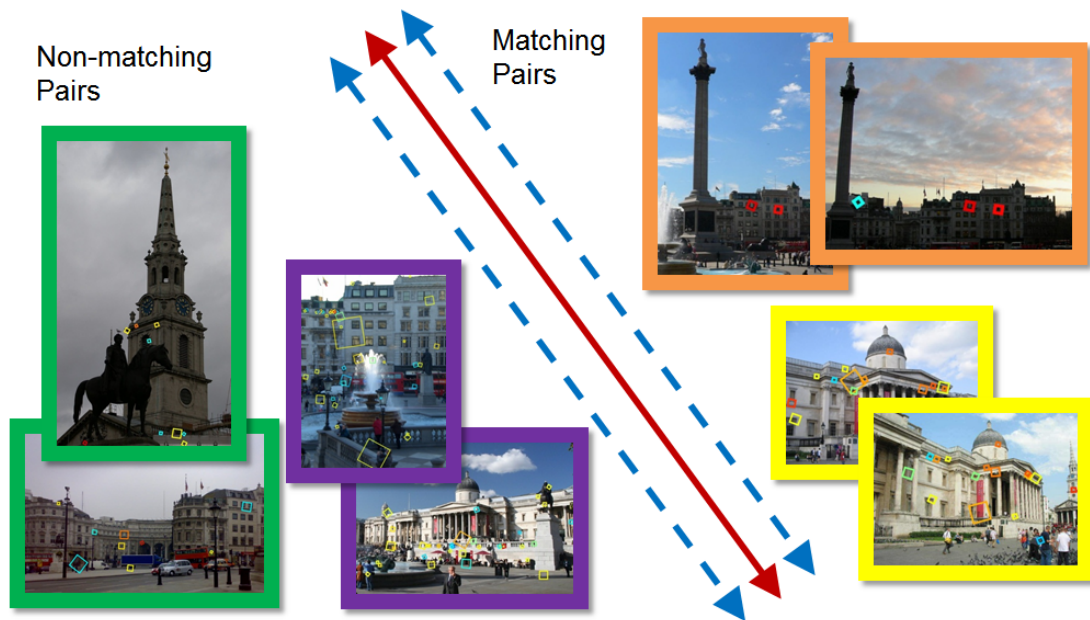


Figure 2.1: **Training an SVM classifier with positive and negative image pairs.** Although each image pair (images with borders of the same color) shown above contain images with many common visual words (shown as boxes with same colors each pair), some pairs are true matches (top right), while others are false matches (bottom left). Accordingly, some visual words are more discriminative (or confusing) than others. Our goal is to learn a weighting of different visual words to better predict matching and non-matching image pairs. This weighting is shown here as a max-margin separating hyperplane. The images shown above are from the **Trafalgar** dataset.



Figure 2.2: **Two example visual words with different discriminative power.** The three images on the left contain a common visual word (in green), which is highly weighted by our learned model. In contrast, the three images on the right also share a common visual word (in red), but do not match; this word is given low weight.

cess of discovering the structure of the image graph; as it attempts to match pairs of input images, the results are used as training data to learn a model specific for that dataset. Motivating our approach is the observation that some visual words are inherently more reliable for measuring image similarity than others, and that these good features seem to be specific to a given dataset (e.g., images of Times Square). For example, some visual words might be more stable across viewpoint and illumination, or less sensitive to quantization errors, than others (Figure 2.2). This suggests that if each visual word is correctly weighted, then our ability to predict whether two images match can improve. While there are many unsupervised ways to define such weights—e.g., *tf-idf* weighting [51], burstiness [22], co-occurrence measures [10]—we explore the use of supervised learning to find a good set of weights, given example pairs of matching and non-matching image pairs from an image set. Unlike prior heuristic approaches, our method is free to leverage whatever structure is present in the data to learn to separate matching image pairs from non-matching pairs.

Given a collection of images (represented as BoW histograms) of a place, our method starts with an unsupervised similarity measure (e.g., *tf-idf*) and automatically generates training data by first finding a small number of image pairs with high similarity, then applying relatively expensive feature matching and verification steps on these pairs. This results in both positive image pairs (successful matches) and negative pairs (unsuccessful matches). We then use discriminative learning (e.g., SVMs), to learn a new similarity measure on features derived from these example image pairs, by posing this as a linear classification problem. Unlike many classification problems, these features are formed from image *pairs*, rather than individual images, as illustrated in Figure 2.1. This process iterates, alternating between proposing more images to match, and learn-

ing a better similarity measure. We show that, even with very small amounts of training data, our learned models consistently outperform recent unsupervised techniques. Moreover, the overhead of learning is quite low; the linear SVMs we use are extremely efficient to compute, even when using a vocabulary of 1M visual words.

Our contributions are two-fold. First, we propose a fast, simple method for using discriminative learning to classify image pairs in large-scale matching problems, showing significant improvement over state-of-the-art unsupervised methods; we also show that a modified form of regularization, as well as drawing negative training examples from unrelated datasets, can improve our learned models. Second, we propose a new iterative image matching method, based on this learning approach, that can reduce the amount of time needed to find matches in large image sets by a factor of more than two on average.

## 2.1 Related Work

**Bag-of-words models and image retrieval.** In BoW models, features such as SIFT [31] are extracted from an input image, then vector-quantized according to a vocabulary of visual words learned from a large set of features (ideally from a related dataset). An image is then represented as a histogram over visual words. Often *tf-idf* weighting is applied to these histograms [51], inspired by techniques from text retrieval. The similarity of an image pair can then be computed as, say, the dot product or histogram intersection of their weighted histograms. BoW models are often used in image retrieval, but are also common in object classification problems [57], where they have been shown to work well combined with discriminative methods. Our problem differs from traditional classification problems in that we seek to classify *pairs* of images of some scene as matching or non-matching, rather than classifying images into categories. This fits our goal of discovering the structure of a large input collection; such collections are often better described as a graph of pairwise connections, rather than a set of discrete categories. While our problem is related to image retrieval, it differs in that the database and query images are one and the same, and we want to discover the structure of the database from scratch—we aren’t matching to a database known in advance. However, we build on methods of computing weights for visual words proposed in the retrieval literature. Many such methods are, like *tf-idf* weighting, unsupervised; Jegou et al. downweight confusing features by modeling burstiness in BoW models [22], while Chum et al. downweight highly correlated sets of visual words (“co-ocsets”) [10]; sparse methods have also been applied to identifying informative features [35]. Although such unsupervised weighting schemes improve retrieval performance, we find that

supervised learning can exploit structure in the data for our image matching problem much more effectively (Figure 2.4). Other methods use a form of supervision, but in a more limited way. For instance, Mikulik et al. create a very fine visual vocabulary and compute a probabilistic model of correlations between similar words [34]; others use image geo-tags [47, 25] to select important features. Probably most related to our work is that of Turcot and Lowe, who also gauge feature importance by performing image matching [54]. However, their approach requires matching every image to  $k$  other database images, then modifying each database vector individually. In contrast, our discriminative learning approach can generalize much more efficiently, learning a useful metric before touching much of the database, which is key to our goal of quickly predicting matching images in large collections. Supervised learning has also been applied to learn better features through non-linear projection of feature descriptors [40]. We instead learn linear classifiers in the high-dimensional BoW feature space.

**Distance metric learning.** Our problem can be considered as treating images as high-dimensional feature vectors, and learning a distance metric between images [56, 48, 16]. We formulate this as learning a classifier over pairs of images, predicting a binary variable (matching/non-matching) for each pair. Although online similarity learning over images has been considered before [7, 4], these formulate the learning problem using triplets of training images; in our problem setting, however, matching or non-matching image *pairs* are more readily available as training data, motivating our formulation. While our automatic training data generation procedure is related to that of [41], we use it in an iterative manner to achieve a different objective than learning topic models.



## 2.2 Our Approach

Given a set of images  $\mathcal{I}$  of a location, our goal is to efficiently compute an image graph on  $\mathcal{I}$  with edges linking overlapping images, by performing detailed SIFT matching and geometric verification on some set of image pairs (edges). Through this matching process, we can determine whether or not the pair overlaps (and which features correspond), by thresholding on the number of geometrically consistent matches. For large collections, we wish to check a small subset of the  $O(n^2)$  possible edges, yielding an approximate graph; hence, we want to intelligently select a subset of edges to match, so as to quickly compute as complete a graph as possible. Our approach seeks an efficient way to predict whether or not a given image pair will match, by learning over time how to classify pairs of images as matching or non-matching. In this section, we formulate this problem as a one of discriminative learning, and propose an iterative approach that alternates between detailed images matching and learning a discriminative model using the matching results.

### 2.2.1 Discriminative Learning of a Classifier for Image Pairs

Consider two images represented as *tf-idf* weighted, sparse, normalized BoW histograms  $a$  and  $b$ , each with dimension  $n$  (with  $n$  equal to, say, 1 million). A typical similarity measure  $\text{sim}(a, b)$  is the cosine similarity, i.e., the dot product  $\text{sim}(a, b) = a^T b$ . A more general way to define a similarity function is  $\text{sim}(a, b) = a^T M b$ , where  $M$  is a symmetric  $n \times n$  matrix. When  $M$  is the identity matrix, this definition reduces to the *tf-idf*-weighted similarity (since  $a$  and  $b$  are “pre-

weighted” with their *tf-idf* weights).<sup>1</sup> At the other extreme, one could learn a full matrix  $M$ ; however, this would be expensive given the high dimensionality of the histograms. In our case, we restrict our method to learning a diagonal matrix  $W$ , which results in a *weighted* dot product of the histograms:  $\text{sim}(a, b) = a^T W b = \sum_i w_i a_i b_i$ , where the  $w_i$ ’s are the diagonal entries of  $W$ . Note that we do not enforce that the  $w_i$ ’s are non-negative, hence  $\text{sim}(a, b)$  not a true metric; nonetheless, we can still use the output as a decision value for prediction. While forcing  $M$  to be diagonal is somewhat limiting, our results suggest that this method still works well in the high-dimensional space of BoW histograms.

Our goal, then, is to learn a weighting  $w_i$  on different dimensions of the visual vocabulary specific to a given dataset; for this, we use the tools of discriminative learning. For a pair of images  $(a, b)$ , we define a feature vector  $x^{a,b}$  as the vector of pair-wise products of corresponding dimensions of  $a$  and  $b$ :  $x_i^{a,b} = a_i b_i$ . Given these features,  $\text{sim}(a, b)$  is simply the dot product of the weight vector  $w$  with the feature vector  $x^{a,b}$ .<sup>2</sup> Given this representation, there is a natural formulation of the learning problem as that of learning a hyperplane—or equivalently a set of weights  $w_i$ —that separate positive (matching) pairs with negative (non-matching) pairs of images. For this problem, we can automatically generate training data by checking if two images match using detailed SIFT matching and geometric verification: pairs  $(a, b)$  that pass become positive training examples  $x^{a,b}$  with label  $y = 1$ ; pairs  $(c, d)$  that do not match become negative training examples  $x^{c,d}$  with label  $y = -1$ . Figure 2.1 illustrates this formulation.

We use  $L_2$ -regularized  $L_2$ -loss SVMs for learning, which in our problem op-

---

<sup>1</sup>We found that such preweighting works better than raw histograms for our learning method.

<sup>2</sup>Other features defined on an image pair could also be used; e.g., defining the features as the element-wise *min* of the two vectors results in a weighted histogram intersection similarity, and creating a feature vector from all  $n^2$  products of word pairs results in learning a full matrix  $M$ .

timize:

$$\min_w \frac{1}{2} w^T w + C \sum_{(a,b) \in S} (\max(0, 1 - y^{a,b} w^T x^{a,b}))^2, \quad (2.1)$$

where  $S$  is the set of training pairs  $(a, b)$ . The output weight vector  $w$  defines a separating hyperplane, but we also interpret it as a similarity measure (a weighted dot product).

While we find that standard linear SVMs work well given sufficient training data, in our setting we start out with no training data, as it is only generated once we start matching images. Given small amounts of training data, standard SVMs can severely overfit the data, performing worse than *tf-idf* weighting. We propose two extensions to address this problem. First, if negative examples from other image collections are available, we find that these can boost the performance when combined with current training data (though positive examples don't seem to help). Second, we utilize a *modified regularization* for SVMs that uses the *tf-idf* weights as a prior. In particular, our modified approach regularizes the weight vector  $w$  to be close to a vector of all ones,  $w_0$  (representing *tf-idf* weighting). To regularize, we substitute  $w$  in the regularization term in (2.1) with  $w - w_0$ , and solve this modified optimization problem. This smoothly transitions between the original *tf-idf* weights and our learned weights, and softly enforces positiveness of the weights, which helps in preventing overfitting and showing significant improvement over both approaches given limited amounts of training data (Section 4.4).

Compared to the feature selection method of Turcot and Lowe [54], we do not rely on explicit correspondence found by SIFT, and instead allow the SVMs



Figure 2.3: **Correct predictions of non-matching pairs.** Due to challenging differences in contrast, illumination and viewpoints, these two image pairs both failed the SIFT matching and verification process, despite exhibiting visual overlap (as well as common visual words, which are marked with boxes of the same color). In contrast, our model is able to correctly highly rank these images, as they happen to have very discriminative visual words (in red). Note that the common visual words may not always imply exact correspondence (e.g., because of repeating patterns).

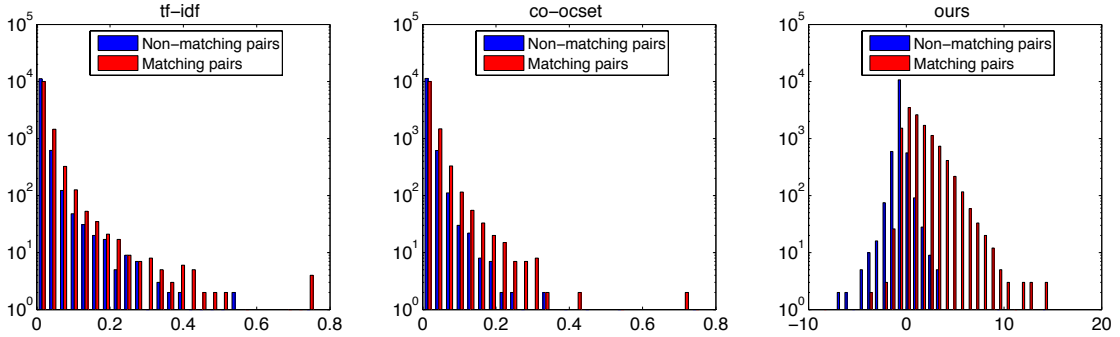


Figure 2.4: **Histograms of matching vs. non-matching testing pairs.** From left to right: histograms of *tf-idf* similarity, co-ocset similarity [10] and our output values ( $\sim 100K$  training pairs) respectively, for matching and non-matching test pairs. Note the log scale on the  $y$ -axis. The test pairs consist of randomly chosen unseen image pairs from the **TateModern** dataset.

to choose the weights as they see fit. Interestingly, although our training data is defined by the output of feature matching, in some cases feature matching fails to identify truly matching image pairs, that our learned model can correctly predict (Figure 2.3). Figure 2.4 demonstrates the predictive power of our method, by comparing histograms of similarities for matching and non-matching pairs generated by our approach and two unsupervised methods (*tf-idf* and the co-ocset method [10]) on the TateModern dataset (Section 4.4). Our method can significantly improve separability of matching and non-matching pairs.

### 2.2.2 Iterative Learning and Matching

In practice, given a new set of images, there is initially no training data to learn from. However, given even a relatively small amount of training data, our algorithm can still boost performance in predicting matching and non-matching image pairs. Thus, we can bootstrap by matching a small subset of pairs, then learning a better similarity measure from the outcome of matching. We start by using the vanilla *tf-idf* weighted image similarities to rank, for each image, all other images. Then our method performs SIFT matching and verification on a small number of highly-ranked image pairs, and trains a linear SVM using the resulting training data. We use the resulting classifier weights to recompute a similarity measure, to re-rank the candidate lists for all images. The system then resumes the image matching process using the new rankings, and repeats.

Given a learned similarity measure, there are many ways to decide the order in which to attempt to match image pairs. We considered two simple strategies: one is to match all image pairs with similarity values above some threshold; the other is to go down re-ranked candidate lists of each images “layer by layer”, matching each image to its most similar candidate in turn. These two strategies have different impacts on the overall system behaviour. In general, the threshold-based strategy generates a higher percentage of true matching pairs out of all pairs tested, while the layer-based strategy “grows” the image graph more uniformly. In our experiments, we adopt the layer-based strategy, as it is less biased towards parts of the image set that are initially ranked as very similar.

## 2.3 Experiments

To evaluate our approach, we collected five image datasets from Flickr, each corresponding to a popular landmark (e.g., Trafalgar Square). and consisting of several thousand images, as summarized in Table 2.1. The sets were chosen so that each contains multiple buildings, views, statues, paintings, etc. (i.e., there is a diversity of views, rather than a single dominant view that would be relatively easy to learn an appearance model for.) Figure 2.5 shows a few example images from the five new datasets we collected.

In addition, each dataset contains images that are not pertinent to the scene itself, such as close-ups of people and photos of water. Note that while these two datasets are similar in sizes with ours, they are also different in that they each consists of several different landmarks. We created a vocabulary of 1M visual words [36] by running hierarchical  $k$ -means on SIFT features from a separate set of images of Rome, used for all 5 datasets. We also tested our method on two standard image retrieval datasets, Oxford5K and Paris [38, 39]; for each we learned specific vocabularies from the database images. We used LIBLINEAR and SVM-LIGHT<sup>3</sup> to learn our SVMs.

### 2.3.1 Performance of Discriminative Learning

First, there are a few key questions about our learning framework that we'd like to answer: How much training data do we need to see an improvement, and how quickly does performance improve with more training data? How much do our two proposed extensions help given limited data? In the limit, given

---

<sup>3</sup><http://www.csie.ntu.edu.tw/~cjlin/liblinear/> , <http://svmlight.joachims.org/>



Trafalgar



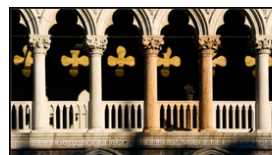
London Eye



Tate Modern



San Marco



Times Square



Figure 2.5: **Some example images from the 5 new datasets we used for evaluation.** Note that in these 5 datasets, there also exist many irrelevant images due to the noisiness inherent in Internet image collections.

large amounts of training data, how good of a similarity function can we learn for a given location? To answer these questions, we devised an experiment testing how well our approach can separate matching and non-matching pairs in each dataset, given different amounts and types of training data. A perfect similarity measure will, for any given image in the dataset, rank all of the matching images in the rest of the dataset above all of the non-matching ones. To measure this, we selected 50 images for each dataset as “queries” and created ground truth by performing SIFT matching and geometric consistency check between these images and all of the other images in that set (for Oxford5K and Paris, however, the standard query images and ground truth are used). We compare our performance with two unsupervised baseline methods: raw *tf-idf* [51] and co-occurrence set (co-ocset) [10] similarities. We measure the quality of the ranking of the rest of the dataset for each query by the average precision (AP), and performance of each model is measured by its mean AP (mAP) over our test set (higher is better).

We trained SVMs with 200, 1,000, and 2,000 randomly sampled image pairs (with no test query images involved in the training), using equal numbers of positive and negative pairs, and determining the regularization parameter  $C$  through cross-validation. To gauge how well our method can perform in the limit, we also trained models with a much larger training set (around 100K training pairs) for each dataset. We also test the effect of our proposed two extensions: modified regularization and adding negative training pairs from an unrelated image set; for the latter, we used the same set of about 1M negative examples from several other datasets.<sup>4</sup>

---

<sup>4</sup>We only test adding such negative data for our 5 Flickr sets, as they share a common vocabulary.



Dataset	#img	tf-idf	co-ocset	200	200+neg	200+mr	
Trafalgar	6981	0.558	0.563	0.620	0.629	0.653	
LondonEye	7047	0.621	0.629	0.586	0.632	0.657	
TateModern	4813	0.712	0.716	0.771	0.793	0.813	
SanMarco	7792	0.577	0.601	0.518	0.535	0.618	
TimesSquare	6426	0.491	0.492	0.410	0.446	0.503	
<b>Average</b>		0.592	0.600	0.581	0.607	0.649	
Oxford5K [38]	5062	0.592	0.608	0.303	-	0.615	
Paris [39]	6412	0.635	0.636	0.505	-	0.652	
<b>Average</b>		0.613	0.622	0.404	-	0.633	

Dataset	1000	1000+neg	1000+mr	2000	2000+neg	2000+mr	~100K
Trafalgar	0.689	0.703	0.698	0.719	0.733	0.725	0.794
LondonEye	0.650	0.676	0.677	0.673	0.694	0.687	0.783
TateModern	0.828	0.835	0.836	0.839	0.851	0.846	0.884
SanMarco	0.606	0.633	0.636	0.637	0.658	0.658	0.766
TimesSquare	0.474	0.535	0.511	0.498	0.563	0.518	0.617
<b>Average</b>	0.650	0.676	0.672	0.673	0.700	0.687	0.769
Oxford5K [38]	0.354	-	0.626	0.397	-	0.629	0.655
Paris [39]	0.620	-	0.668	0.632	-	0.676	0.695
<b>Average</b>	0.487	-	0.647	0.514	-	0.652	0.675

Table 2.1: **mAP performance of models trained with various training data sizes.** The baselines are the mAP scores for rankings using *tf-idf* similarity and co-ocset similarity [10]. Columns marked with **200**, **1000**, **2000**, and **~100K** show the performance of models trained with corresponding number of examples. *N+neg* corresponds to models trained with the same set of *N* examples combined with large amounts of negative examples from other datasets; *N+mr* denotes models trained with the same set of *N* examples using our modified regularization.

The results are shown in Table 2.1. For our 5 Flickr datasets, our models trained with 200 examples (with standard regularization) are slightly worse on average than *tf-idf* similarity, probably due to overfitting to insufficient training data, but both extensions of our approach prove effective in dealing with this issue, each exceeding our baselines on average. Unsupervised co-ocset similarity also shows improvement over *tf-idf* similarity, but our models consistently outperform both; even our unmodified method trained with 1,000 examples

outperforms both baselines, and with 2,000 examples the mAP improves even further. The performance of models trained with  $\sim 100\text{K}$  examples jumps by a significant margin, illustrating the large potential improvement of our discriminative learning approach over time. Note that 100K examples is still a small fraction of the total number of possible pairs in each set (e.g., the **Trafalgar** dataset, with 6,981 images, has over 24M image pairs). Comparing our two extensions, we find that the improvement by modified regularization is more significant when there is very little training data (e.g. 200 examples), while adding unrelated negative examples gives a larger improvement when more data is available (e.g. 2,000 examples). Because Oxford5K and Paris each encompass several disparate landmarks, they require more training data, and hence modified regularization is essential for these two datasets. With modified regularization, models trained with only 200 examples outperform the baselines. We also tested with much lower amounts of training data; we found that with as few as 20 training examples, our method can consistently outperform both baselines in all datasets (Table 2.2).

The mAP score above is also used in image retrieval, though we emphasize that we address a different problem in that we seek to discover the connectivity of an entire image set. Our method focuses on learning similarity measures, and as such is orthogonal to other popular methods for improving image retrieval, such as query expansion [11], Hamming embedding [23], or using better feature detectors than the DoG detector. Hence, while our baseline is not as good as that achieved in [38] (e.g. 0.618 for Oxford5K), our method could be combined with others to achieve even better performance.

We can also evaluate our learned models as classifiers (e.g., if we wanted to

Dataset	# Pairs	Original Reg.	Modified Reg.	tf-idf	co-ocset
Trafalgar	20	0.388	0.598	0.558	0.563
	40	0.485	0.616		
	200	0.620	0.653		
	1,000	0.689	0.698		
	2,000	0.719	0.725		
LondonEye	20	0.412	0.631	0.621	0.629
	40	0.536	0.646		
	200	0.586	0.657		
	1,000	0.650	0.677		
	2,000	0.673	0.687		
TateModern	20	0.564	0.776	0.712	0.716
	40	0.699	0.788		
	200	0.771	0.813		
	1,000	0.828	0.836		
	2,000	0.839	0.846		
SanMarco	20	0.197	0.609	0.577	0.601
	40	0.306	0.611		
	200	0.518	0.618		
	1,000	0.606	0.636		
	2,000	0.637	0.658		
TimesSquare	20	0.247	0.497	0.491	0.492
	40	0.305	0.497		
	200	0.410	0.503		
	1,000	0.474	0.511		
	2,000	0.498	0.518		
Oxford5K	20	0.128	0.598	0.592	0.608
	40	0.151	0.608		
	200	0.303	0.615		
	1,000	0.354	0.626		
	2,000	0.397	0.629		
Paris	20	0.228	0.641	0.635	0.636
	40	0.292	0.643		
	200	0.505	0.652		
	1,000	0.620	0.668		
	2,000	0.632	0.676		
Average	20	0.309	0.621	0.598	0.606
	40	0.396	0.630		
	200	0.530	0.644		
	1,000	0.603	0.664		
	2,000	0.628	0.677		

Table 2.2: **Improved mAP on all 7 datasets using modified regularization with limited training examples.** This table shows the performance of our learning method with the standard SVM regularization, with our modified regularization, and the baselines, *tf-idf* similarity and “co-ocset” similarity, in the task of ranking. “# Pairs” shows the number of training image pairs used for training corresponding models. The modified regularization performs better than either other method when small amounts of training data are available, though the performance gap between the two regularization approaches shrinks, as expected, when more training data is used.

make a forced prediction of all edges in the image graph at some point during the matching process). For each dataset we measured the classification accuracies on a held-out set of test image pairs, with equal numbers of matching and non-matching pairs. With our fully trained models ( $\sim 100\text{K}$  examples), we observed an average accuracy of  $90.4(\pm 2.9)\%$ .

### 2.3.2 System Evaluation

While the experiment above illustrates that our learning framework can yield better similarity measures, how well does our iterative matching system work in practice? The training pairs we get while matching will be different from the random ones selected above. Hence, we also evaluate the performance of our iterative matching system as a whole by running it on the datasets described above. As a reminder, our algorithm matches images in rounds, initially using *tf-idf* similarity to rank candidate pairs for matching, but learning a better model over time. Learning initially takes place once a certain number  $N$  of image pairs have been processed. We observe that the margin of performance improvement decreases as the number of training instances (rounds) increases, so at each round we match more image pairs than last round by a factor of  $\beta$  before training. This increases overall efficiency, as learning and re-ranking take time. In our experiments, we use  $\beta = 1.5$  and  $N = 2000$ . We compare to a baseline system that does not rerank image pairs, and simply processes each image’s most similar candidates in the order computed by *tf-idf* similarity. This mimics current similarity-based large scale image matching methods [1, 41]. We terminate when  $\geq 40$  candidates are processed for each image. For this experiment, *efficiency* is the key metric—how quickly can we find matches, and what percent

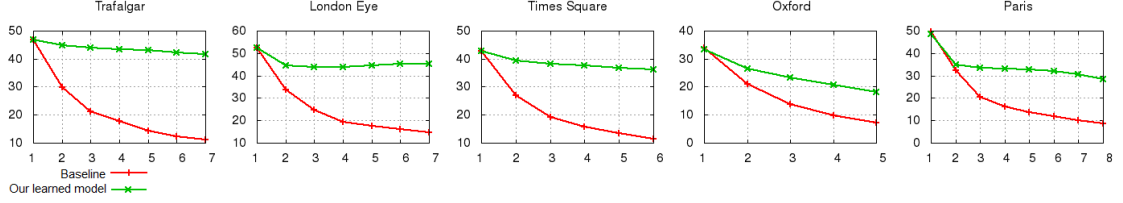


Figure 2.6: **Per-round *match success rates* for five datasets.** The  $x$ -axis is the round number, and the  $y$ -axis is the percentage of image pairs tested so far found to be true matches. Since we use *tf-idf* similarity in the first round, the corresponding percentages are the same for that round.

of the image pairs we try turn out to be true matches (meaning we didn’t waste effort matching them)? Hence, we evaluate performance after each round of matching using the percentage of image pairs tested so far that were found to be true matches. A higher *match success rate* indicates better efficiency.

Match success rate over time for five datasets are shown in Figure 2.6; the other datasets show a similar trend. Aside from the initial round (where we use *tf-idf* similarity), our system significantly improves the match success rate. For instance, for the Trafalgar dataset, after seven rounds of matching, our method has a success rate of over 40%, while the baseline method has a success rate of just over 10%. We also found the mAP metric used in Section 2.3.1 also improves gradually over time.

We found that the overhead of training and re-ranking between rounds is much less than the time spent on image matching. For the Oxford5K dataset, our measured CPU time for matching was 2,621 minutes, while training and re-ranking took 17 and 118 minutes respectively (0.66% and 4.49% of image matching time). To obtain 7,000 matching image pairs, the *tf-idf* similarity-based image matching method checked over 90K image pairs ( $\geq 1525$  CPU minutes) while our approach checked fewer than 31K ( $< 707$  CPU minutes including training

and re-ranking overhead), more than a factor of two improvement in efficiency. Figure 2.7 shows the number of successful matches found as a function of total CPU running time for all 7 datasets, for our image matching system and for the baseline based on raw *tf-idf* similarities. We observe that to obtain the same number of true matches, the CPU time required by the *tf-idf* similarity-based approach is often more than a factor of two times higher than our approach (except for the first couple of rounds of learning and matching, where the overhead of our method is comparatively high).

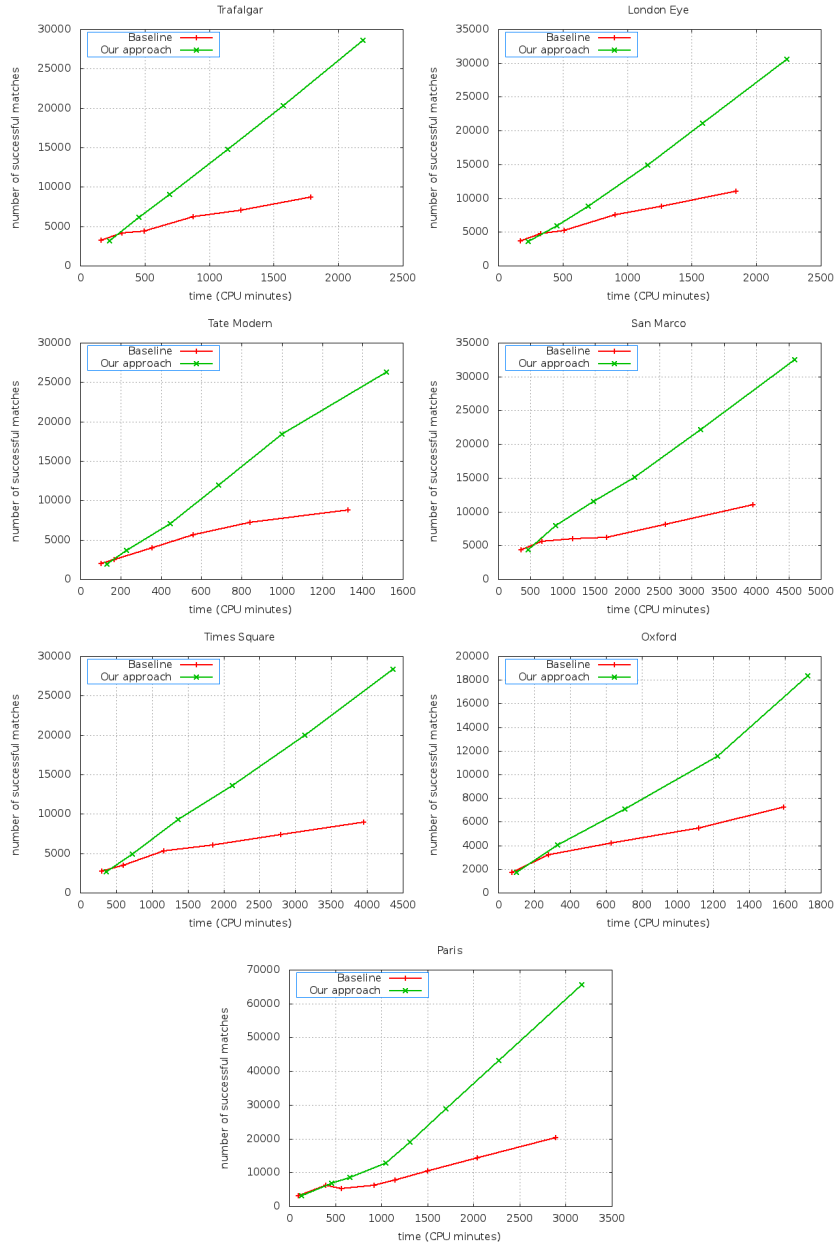


Figure 2.7: **Number of successful matches as a function of CPU time.**

This figure plots the numbers of true matches found as a function of CPU time spent, for both our approach and the baseline, for each dataset. After each round (shown by the data points above), the time for our approach is measured as the sum of image matching time, training time and re-ranking time, while that of baseline only includes image matching time. Our approach shows significant improvement in terms of efficiency in discovering true matching image pairs.

## CHAPTER 3

# GRAPH-BASED DISCRIMINATIVE LEARNING FOR LOCATION RECOGNITION

What is a place? In this chapter, we consider this question in the context of the location recognition problem—determining where an image was taken—in particular, in choosing a good *representation* for places we wish to recognize. There is no single definition for what it means to be a place, and, accordingly, a wide variety of representations for places have been used in the literature: Are places, for instance, a set of discrete landmarks, each represented by a set of images? [59, 28] Are places latitude and longitude coordinates, represented with a set of geotagged images? [20] Should places be represented with 3D geometry, from which we can estimate an image’s location from a continuum of possible viewpoints? [29, 45, 30] Is a place a set of representative visual elements? [13] This question of *representation* has analogues in more general object recognition problems, where many approaches regard objects as belonging to pre-defined categories (cars, planes, bottles, etc.), but other work represents objects more implicitly as structural relations between images, encoded as a graph (as in the Visual Memex [32]).

Inspired by this latter work, this chapter address the location recognition problem by representing places as *graphs* encoding relations between images, and explore how this representation can aid in visual recognition. In our case, our graphs represent visual overlap between images—nodes correspond to images, and edges to overlapping, geometrically consistent image pairs—leveraging recent work on automatically building image graphs (and 3D models) from large-scale image collections [1, 15, 12], and techniques introduced in



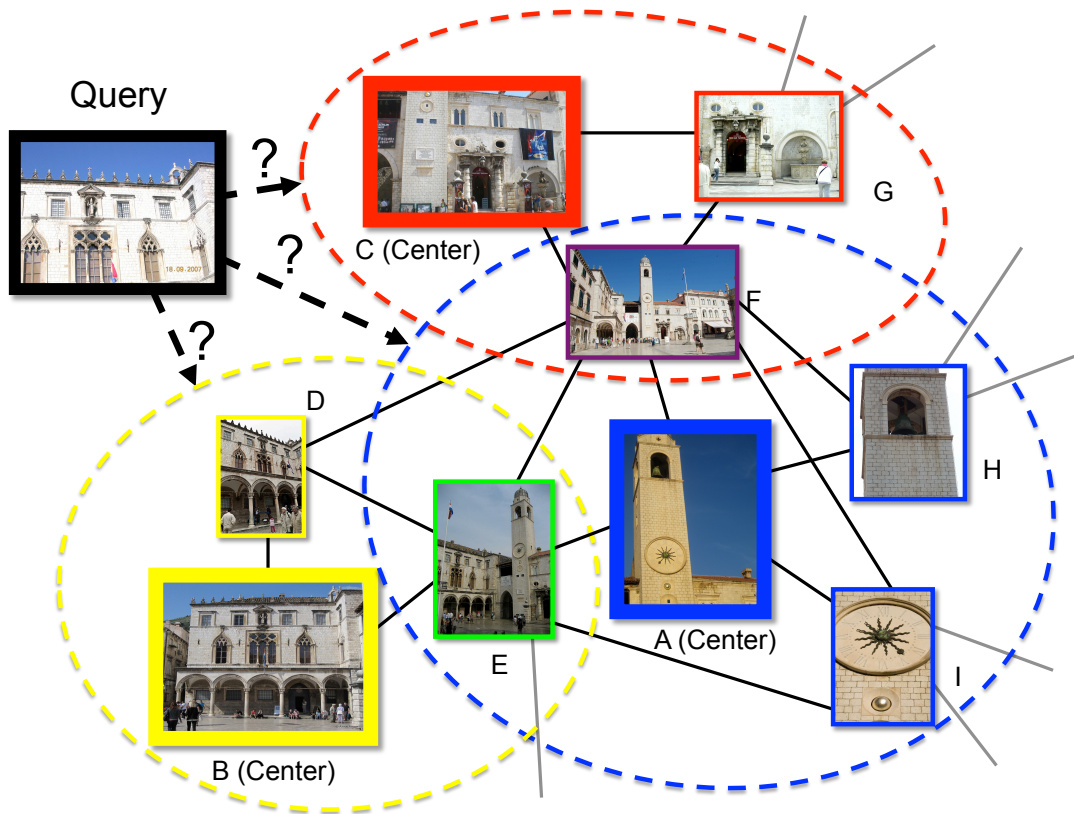


Figure 3.1: **A region of an example image graph with three neighborhoods defined by representative images A, B and C.** Nodes in this graph are images, and edges connect visually overlapping images. Our method uses the graph to find a set of representative neighborhoods (inside the dotted circles above) that cover the graph, and learns a local distance function for each neighborhood. These distance functions are used to connect a new query image (left) to the rest of the graph and hence recognize its location. Given a query image, we match to the graph using these learned neighborhood models, rather than considering database images individually. Each neighborhood has its own distinctive features, and our goal is to learn and use them to aid recognition.

Chapter 2 [5].

An example image graph for photos of the town of Dubrovnik is shown in Figure 3.1. Given an image graph, our goal is to take a query image and “plug it in” to the graph in the right place, in effect recognizing its location. The idea is that the structure inherent in these graphs encodes much richer information than the set of database images alone, and that utilizing this structural information can result in better recognition methods.

We make use of this structural information in a bag-of-words-based location recognition framework, in which we take a query image, retrieve similar images in the database, and perform detailed matching to verify each retrieved image until a match is found. While others have used image graphs in various settings before (especially in 3D reconstruction), our main contribution is to introduce two new ways to exploit the graph’s structure in recognition. First, we **build local models** of what it means to be similar to each neighborhood of the graph (Figure 3.1). To do so, we use the graph’s structure to define sets of images that are similar, and sets that are different, and use discriminative learning techniques to compute local distance functions tuned to specific parts of the graph, and hence, to specific parts of the location. Second, we use the connectivity structure of the graph to **encourage diversity** in the set of results, using a probabilistic algorithm to retrieve a shortlist of similar images that are more likely to have at least one match. We show that our graph-based approach results in improvements over bag-of-words retrieval methods, and yields performance that is closer to more expensive direct feature matching techniques on existing location recognition datasets.

### 3.1 Related Work

**Image retrieval and supervised learning.** As with other location recognition approaches [47, 21, 25, 46], our work uses an image-retrieval-based framework using a bag-of-words model for a database of images. However, our goal is not retrieval per se—i.e., we do not seek to retrieve all related instances of a query image—but instead recognition, where we aim to determine where an image was taken—for which a single correctly retrieved database image can be sufficient).

Our work uses supervised learning to improve on such methods. Prior work has also used various forms of supervision to improve bag-of-words-style methods for both retrieval and recognition. One type of supervision is based on *geolocation*; images that are physically close—on the same street, say—should also be closer according to some notion of image distance than images across the city or the globe. Geolocation cues have been used to reweight different visual words based on their geographic frequency [47, 25], or to find patches that are discriminative for different cities or neighborhoods [13]. Other methods rely on image matching to identify good features, as we do. Turcot and Lowe [54] perform feature matching on database images to find reliable features. Arandjelovic and Zisserman propose *discriminative query expansion* in which a per-query-image distance metric is learned based on feedback from image retrieval [2]. Mikulik et al. use image matches to compute global correlations between visual words [34]. In contrast, we use discriminative learning to learn a set of *local* distance metrics for the database as a pre-process (rather than at query time), leveraging the known graph structure of the database images.

**Representing places.** Places in computer vision are often represented as bags of images; for instance, the Eiffel Tower can be described as a collection of photos showing that landmark [59]. However, many other representations of places have been explored. Some methods use *iconic images* to represent sets of images taken from very similar viewpoints [27, 24], or otherwise select a set of representative views [21]. Other approaches use 3D point clouds, derived from structure from motion, and augmented with appearance information, as a richer geometric representation of a place [29, 44]. Closest to our approach are methods that explicitly construct and exploit image graphs. For instance, Torii et al. download Google Streetview images to form a recognition database, and make use of the underlying Street View image network. In their approach, they take linear combinations of neighboring images (in bag-of-words space) to more accurately recognize the continuum of possible viewpoints of a city [53]. Li et al. use a visibility graph connecting images and 3D points in a structure-from-motion model to reason about point co-occurrence for location recognition [30].

A main contribution of our approach is to combine the power of discriminative learning methods with the rich structural information in an image graph, in order to learn a better underlying representation from the image database, and to better analyze the retrieval results at query time.

## 3.2 Graph-based Location Recognition

This section describes our problem setting and preliminaries, and provides an overview of our method. In our work, we take as our database a set of images  $\mathcal{I}$  of a place, or set of places; the places we consider generally span a large area, such as an entire city, or multiple landmarks across a wide region. From this set of images, we first compute two types of information: **appearance** information for each image (in our work we use  $L_2$ -normalized bag-of-words histograms [51]), and **connectivity** information represented as an image graph  $\mathcal{G}$  on the set of images  $\mathcal{I}$ . The graph  $\mathcal{G}$  contains a node for each image  $a \in \mathcal{I}$ , and an edge  $(a, b)$  connecting pairs of visually overlapping, geometrically consistent image pairs. Our goal is to use this structural information to help us quickly and accurately take a new query image and predict which part of the graph it is connected to, from there determine its fine-grained location within the place.

To achieve this goal, as in many image retrieval methods, we use the query to retrieve a shortlist of similar database images based on bag-of-words similarity, and perform detailed matching and geometric verification on the top few matches to (a) verify whether the match is correct and (b) determine the location of the image. Ideally, a correctly matching database image will appear near the beginning of the shortlist, if not in the top spot; because our goal is recognition, rather than retrieving all matching instances in the database, we can stop as soon as we find a correct match. Towards that end, our contribution is a method that improves on the often noisy raw bag-of-words similarity measure by leveraging the graph in two ways: (1) we discriminatively learn local distance functions on neighborhoods of the image graph (Section 3.3), and (2) we use the graph to generate a ranked list that encourages more diverse results

(Section 3.4).

First, we describe how we compute the image graph  $\mathcal{G}$  from the set of images  $\mathcal{I}$ .

**Image Graphs.** Our method starts by constructing an image graph  $\mathcal{G}$  from the image database using a standard image matching pipeline [1]. In short, we extract features from each image, and perform pairwise feature matching on a set of candidate image pairs proposed using bag-of-words image similarity. For each such pair, we find nearest neighbor features matches from one image to the other, prune these matches using Lowe’s ratio test, and perform RANSAC-based geometric verification on the remaining features to compute a set of inlier matches (note that we only perform feature matching in one direction). Our method does not use these matches directly, but instead records the number of inlier feature matches for each image pair. Note that we could further improve the quality of these matches by running structure from motion to obtain a point cloud and a refined set of image correspondences, though this is not required by our method.

For each image pair  $(a, b)$  with sufficient inlier matches (we use a threshold of 12 in our experiments), we create an edge in our graph  $\mathcal{G}$ . We also save this number of inliers, denoted  $N(a, b)$ , for each image pair, and use this measure to derive edge weights for the graph. In our experience, the graphs we compute have very few false positive edges—almost all of the matching pairs are correct—though there may be edges missing from the graph because we do not exhaustively test all possible edges.

Our algorithm also weights each edge in the graph with an estimate of the

strength of the visual connection between the two images; later stages of our algorithm threshold edges by their weights. While there are many potential ways to define these weights, we found that an effective definition is one related to the idea of a *Jaccard index*. We define this weight as:

$$J(a, b) = \frac{N(a, b)}{N(a) + N(b) - N(a, b)}, \quad (3.1)$$

where  $N(a)$  and  $N(b)$  denote the number of features in  $a$  and  $b$  respectively that were matched to any other image (which we refer to as their *matchable* features).<sup>1</sup> In other words,  $J(a, b)$  measures the similarity of the two images as the number of features  $N(a, b)$  they have in common, normalized by the union of their matchable feature sets. This measure ranges from 0 to 1; 0 if no overlap, and 1 if every feature was matched between the two images. This normalization reduces bias towards images with large numbers of features. Alternatively, one could also use the raw number of features matches  $N(a, b)$  as the edge weights, but we found in our experiments that the “normalized” weights defined by the Jaccard index work better in practice.

---

<sup>1</sup>Note that feature detectors such as SIFT often detect unstable features in an images that are not matched to features in any other image in the collection.

### 3.3 Graph-based Discriminative Learning

How can we use the information encoded in the image graph to better recognize the location of a query image? One key piece of information provided by the connectivity of the image graph is a notion of similarity—i.e., we know which images are expected to be similar (connected pairs) and which are not (disconnected pairs), according to some desired distance metric. Hence, a natural way to approach our problem is as one of learning a distance metric between pairs of images. We have considered several possible ways to learn such a distance metric using the image graph. For example, one could take all the connected pairs in the graph to be positive example pairs, and all other pairs as negative example pairs, and learn a single, **global** image distance metric for a specific image graph, e.g., using the machinery of SVMs [5]. At the other extreme, one could learn a **local** distance metric for each image in the database, similar to how Exemplar-SVMs have been used for object detection [33].

We tried both of these approaches, but found that, in practice, we achieve better performance with an approach that balances these two extremes. In particular, we divide the graph into a set of overlapping, representative subgraphs, and learn a separate distance metric for each of these representative subgraphs (we will also refer to each such subgraph as a “neighborhood,” because we select each subgraph as the neighborhood of a particular exemplar image). Our method can thus adaptively learn the appearance of different parts of the scene, but chooses these parts effectively so as to result in a stable set of learning problems. These learned distance metrics are then used at query time to determine to which neighborhood a query image belongs. Our approach consists of the following steps:



### At Training Time

1. Compute a covering of the graph with a set of overlapping subgraphs.
2. Learn and calibrate a distance metric for each subgraph.

### At Query Time

3. Use the models in Step 2 to compute the distance from a query image to each database image, and generate a ranked shortlist of possible image matches.
4. Perform detailed matching and geometric verification with the top database images in the shortlist, until a successful true image match is found.
5. Optionally use this match to further refine the query image location, e.g., through pose estimation.

We now describe each step in more detail. Later, in Section 3.4, we discuss how we further improve Step 3 by reranking the shortlist to encourage diversity, based on the structure of the graph.

**Step 1: Selecting representative neighborhoods.** We begin by selecting a set of representative subgraphs that *cover* the entire graph. Once we have selected these subgraphs, we will learn a local similarity function for each subgraph, using the images in the subgraph as positive examples, and other, unrelated images in the graph as negative examples. What makes a good subgraph for learning such local distance functions? We want each subgraph to contain images that are broadly similar, so that our learning problem has a set of positive example images that are relatively compact in appearance space, and can be explained with a simple model. On the other hand, we also want as many positive examples as possible, so that our models have enough data from which to gen-

eralize. Finally, we want our subgraphs to completely cover the graph (i.e., each node is in at least one subgraph), so that we can build models that apply to any image of the location modeled in the database.

Based on these criteria, our algorithm covers the graph by selecting a set of representative *exemplar* images, and defining their (immediate) neighborhoods as subgraphs in a graph cover, as illustrated in Figure 3.1. Formulated this way, the covering problem becomes one of selecting a set of representative images that form a *dominating set* of the graph. For a graph  $\mathcal{G}$ , and a set of exemplar images  $C$ , we say an image  $a \in \mathcal{I}$  is covered by  $C$  if either  $a \in C$ , or  $a$  is adjacent to an image in  $C$ . If  $C$  covers all nodes, then  $C$  is a dominating set. For efficiency and robustness of learning, we would like  $C$  to be as small as possible, and conversely the neighborhood of each node in  $C$  to be as large as possible. Hence, we seek a *minimum* dominating set. Such sets have been used before for 3D reconstruction [19]; here we use them to define a set of neighborhoods for which we will compute learned distance functions.

Finding an exact minimum dominating set is an NP-complete problem, but this problem has several approximation algorithms. In our case, we use a simple greedy algorithm to find an approximate solution [18]. Starting with the empty set, we iteratively add one image at a time to the set of selected exemplars,  $C$ . At each iteration we maintain, for each candidate image  $a$ , the number of as-yet uncovered images  $d_a$  that  $a$  covers, and choose the candidate that maximizes  $d_a$  to add to the selected set. This process repeats until all images are covered. Figure 3.2 shows an example image graph for the Dubrovnik dataset [29] and the exemplar images selected by this greedy algorithm (shown as red nodes). Figure 3.3 shows some example neighborhoods found in this way, each of which

is sorted by the Jaccard index compared to the center image. Note that the Jaccard index values tend to be lower than one might expect, i.e. the number of points each pair of image share is often a small fraction (e.g., 0.2) of the total number of points visible in each image. This is likely due to the noisy process of feature matching as well as the conservative outlier rejection used by structure from motion. In this way, we seek to use the image graph to give us a clean separation between sub-locations, as well as sufficient training examples for each location.

**Step 2a: Discriminative learning on neighborhoods.** For each neighborhood selected in Step 1, the next step is to learn a distance metric for comparing new images to each neighborhood. We treat this as a classification problem, i.e., we seek to learn a classifier for each neighborhood that will take a new image, and classify it as belonging to that neighborhood or not. We learn these classifiers using standard linear SVMs on bag-of-words (BoW) histograms, one for each neighborhood, and calibrate the set of SVMs as described in Step 2b. At query time, these classifiers are used to define a set of similarity functions for ranking the database images given a query image. This use of classifiers for ranking has found many applications in vision and machine learning, for instance in image retrieval using local distance functions [17] or Exemplar-SVMs [49]. Note that while we use the term “exemplar” for the central image of each neighborhood, we use more than the single exemplar image as a positive example for that neighborhood; multiple images from each neighborhood are used as positives for training a classifier for recognizing new images that belong to that neighborhood.

To learn these classifiers, for each exemplar image  $c \in C$  defining a neighbor-

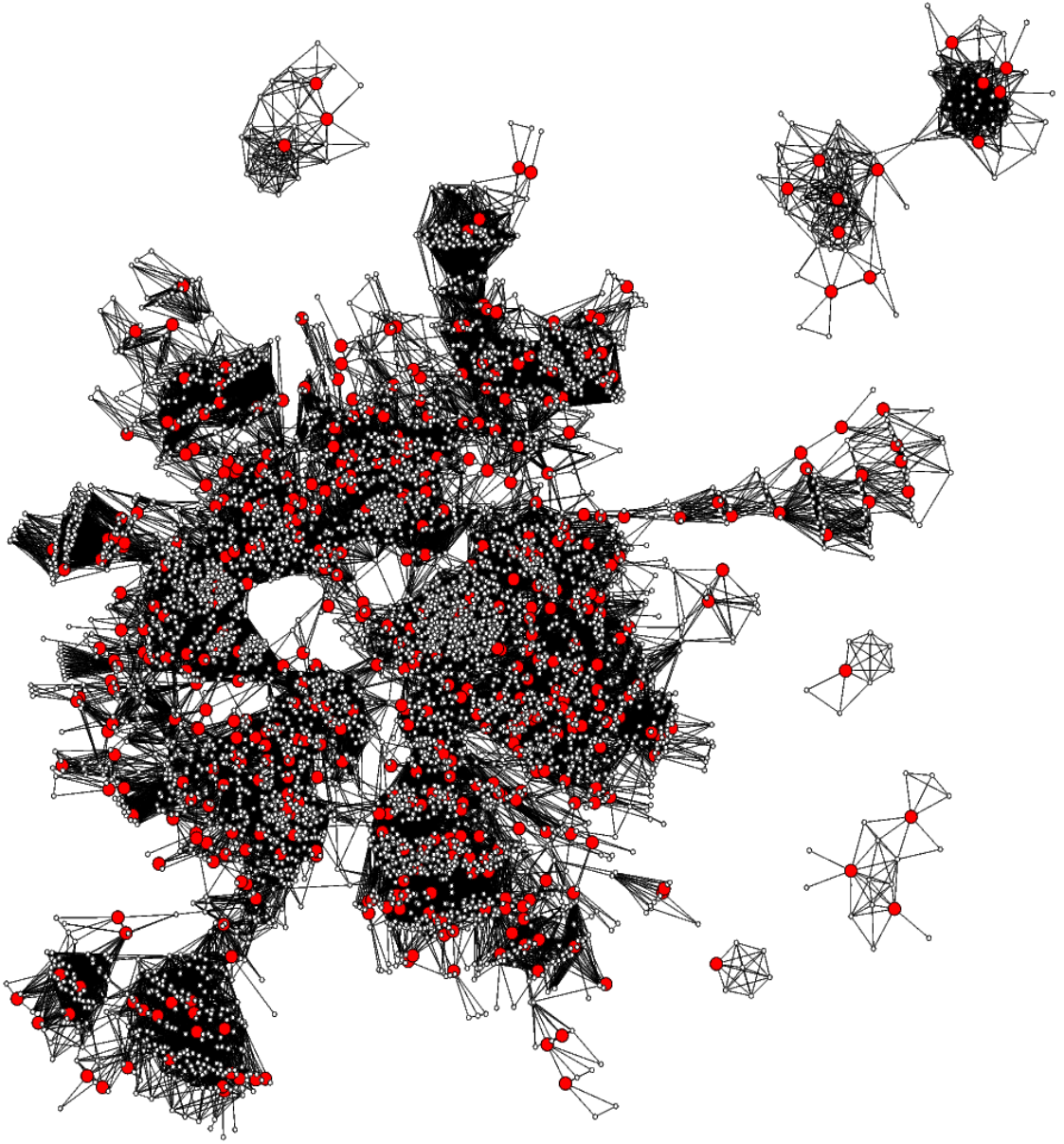


Figure 3.2: **Image matching graph for the Dubrovnik dataset and selected exemplar nodes.** This graph contains 6,844 images (white dots); the large, red nodes denote representative exemplar images selected by our covering algorithm (478 images in total). Although the set of representative images is much smaller than the entire collection, their neighborhoods cover the matching graph. For each neighborhood, we learn a classifier for determining whether a new image belongs to that neighborhood. (Note that the layout of the graph is based on its structure, and not any underlying image locations.)

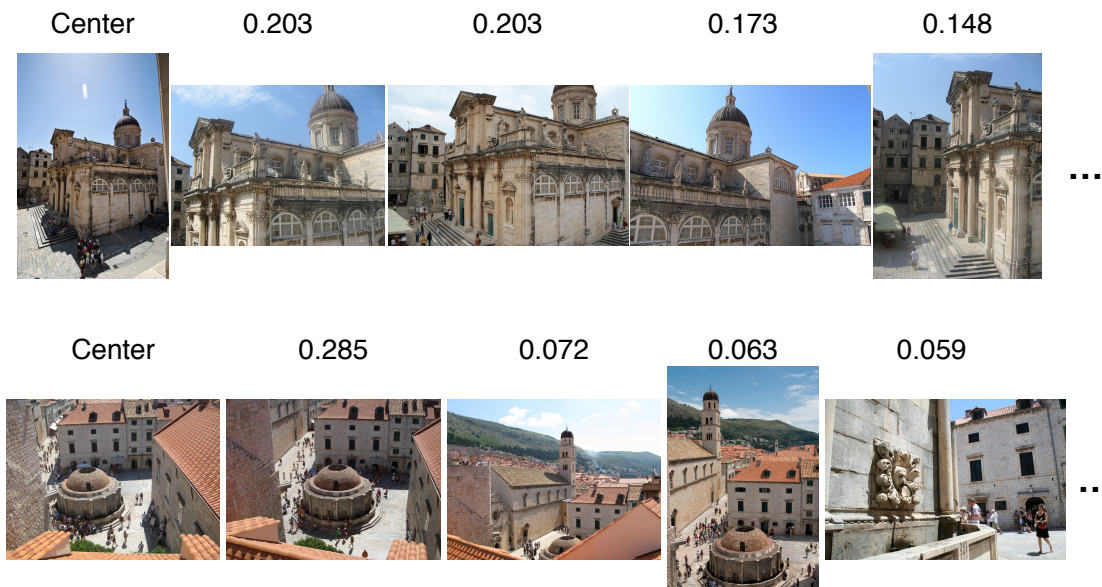


Figure 3.3: **Two example neighborhoods produced by our covering algorithm.** Each row shows an example neighborhood. The left-most image in each row is the exemplar (center) image for each neighborhood. The remaining images are sample members of that neighborhood, sorted in decreasing order by their edge weights (Jaccard index) with the center image. We have found that this way of representing locations—as collections of overlapping image neighborhoods—is a natural one, allowing us to use discriminative learning to learn the most representative features to best identify the location of a query image, adapting to each neighborhood.

hood, we must define a set of positive and negative example images as training data for the SVM. To define the positive set, we simply use the images in the neighborhood of  $c$ . In particular, we found that thresholding the edges in the graph by their weight—effectively applying a stricter definition of connectivity, and yielding more compact neighborhoods—yields better classifiers than using all edges found by the image matching procedure. In other words, to define the positive set for exemplar image  $c$ , we select all connected images  $a$  such that  $W(a, c) > \tau$ , where  $W(a, c)$  is the weight of edge  $(a, c)$  (e.g., the Jaccard index  $J(a, c)$ ), and  $\tau$  is a threshold.

To define the negative set for the neighborhood around exemplar  $c$ , we want to minimize the chance of including false negatives in the training set. Hence we use the original graph, as opposed to the thresholded graph, to define the negatives. More specifically, we select all images  $b$  such that  $(b, c) \notin \mathcal{G}$  (we define  $W(b, c) = 0$  in this case). In this way, the image graph  $\mathcal{G}$  provides the supervision necessary to define positives and negatives for learning, just as geotags have been used as a supervisory cue for discriminative location recognition in previous work [47, 25].

Given the training data for each neighborhood, we learn a linear SVM to separate images which belong to the neighborhood from images which do not. We use tf-idf weighted,  $L_2$  normalized bag-of-words histograms for each image as features.<sup>2</sup> We randomly split the training data into training and validation subsets for parameter selection in training the SVM, and use both for fitting a logistic regressor in Step 2b (more details in Section 3.5.2). For each exemplar image  $c$  and its neighborhood, the result of training is an SVM weight vector  $\mathbf{w}_c$  and a bias term  $b_c$ . Given a new query image, represented as a bag-of-words vector  $\mathbf{q}$ , we can compute the decision value  $\mathbf{w}_c \cdot \mathbf{q} + b_c$  for each neighborhood defined by exemplar image  $c$ , which we will use to rank the neighborhoods. Figure 3.4 shows visualizations of a few results of this training procedure. Each image is overlaid with features assigned highly positive or negative weights by our learned SVM models. (Note that features are rarely assigned negative weights, but such features are potentially discriminative as usually belonging to other locations.) One interpretation of the highly positive features is that they are the ones that are both discriminative of their particular neighborhood,

---

<sup>2</sup>Note that the tf-idf weighting of the bag-of-words histograms could be seen as redundant for the purposes of learning an SVM since the SVM is itself learning a per-word weight. However, we found such “pre-weighting”—i.e., using tf-idf weighted histograms as opposed to raw histograms as feature vectors—to be useful in practice, perhaps as a form of regularization.

and consistent within their own neighborhood. The beauty of the image graph is that its structure gives us an elegant way to learn these weights automatically.

These learned weight vectors can be thought of as defining a similarity between a query and a database neighborhood as a dot product between the query and the weight vector (plus a bias term). Alternatively, at this point, we could simply *replace* our entire database with these set of weight vectors  $w_c$  (plus an extra dimension for the bias term). Ranking the exemplar images by classifier score would be equivalent to computing the dot product of a query image with each weight vector, and sorting by these dot products. (This can be thought of as replacing each exemplar image’s original bag-of-words vector in the database with a new, discriminatively trained bag-of-words vector.) However, we first need to calibrate all of the different SVMs with respect to each other. In addition, we found it important to rank *all* database images (not just the exemplars) for a given query, as described in Step 3 below.

**Step 2b: Calibrating classifier outputs.** Since the classifier for each neighborhood is independently trained, we need to normalize their outputs before comparing them. To do so, we convert the decision value of each SVM classifier into a probability value, using Platt’s method [42] on the entire set of training data. We found that using more data yields better performing logistic regressor, hence unlike SVM training, we use both training and validation sets. For a neighborhood around exemplar  $c$ , and a query image vector  $q$ , we denote the resulting probability value as  $P_c(q)$ . We found this independent calibration procedure to work well in practice.

**Step 3: Generating a ranked list of database images.** For a query image represented as a BoW vector  $q$ , we can now compute a probability value  $P_c(q)$  for  $q$





Figure 3.4: **Visualization of learned SVM weights on two example neighborhoods.** Here, we show, for two neighborhoods, several example images from each neighborhood. The SIFT features shown in each image are color-coded to visualize the weights for their corresponding visual words, as learned through our SVM training process for each neighborhood. The closer to red the color, the higher the corresponding weight is; the closer to blue, the lower the weight is. Most of the features (typically  $> 99\%$ ) are assigned zero weights (and are not shown above), due to the  $L_1$  regularization used for training our SVMs and due to the large vocabulary size (1M in our experiments, see Section 4.4). The features shown here are assigned high positive or negative weights. Note that most features have positive weights, whereas negative weights provides better separation between different neighborhoods. For example, in the top-right image of the neighborhood on the left, negative weights on the other side of the building belong to another neighborhood while features on the front side get highly weighted. In addition, note how repetitive features get mid-range positive weights while discriminative and distinctive features get highest weights.



belonging to the neighborhood of each exemplar image  $c$ . Using these values, it is straightforward to generate a ranked list of the exemplar images  $c \in C$  by sorting by  $P_c(\mathbf{q})$  in decreasing order. However, we found that verifying the query image against exemplar images alone sometimes failed simply because the exemplar images represent a much sparser set of viewpoints than the full graph. Hence, we would like to create a ranked list of *all* database images. To do so, we take the sorted set of neighborhoods given by the probability values, and then we sort the images *within* each neighborhood by their original tf-idf similarity. We then concatenate these per-neighborhood sorted lists; since a database image can appear in multiple overlapping neighborhoods (see Figure 3.1), in the final list it appears only in list of its most highly ranked neighborhood. This results in a ranking of the entire set of database images. Note that we cannot use the learned weights directly here, since each neighborhood corresponds to a single weight vector that does not differentiate members within that neighborhood.

**Step 4: Geometric verification.** Finally, using the ranking of database images from Step 3, we perform feature matching and RANSAC-based geometric verification between the query image and each of the images in the shortlist in turn, until we find a true match. If we have additionally have a 3D structure-from-motion model associated with the database, we can further associate 3D points with matches in the query image, and determine the query image’s camera pose [30]. Otherwise, we can associate the location of the matching database image as the approximate location of the query image. Because feature matching and verification is relatively computationally intensive, the quality of the ranking from Step 3 highly impacts the efficiency of the system—ideally, a correct match will be among the top few matches, if not the first match.

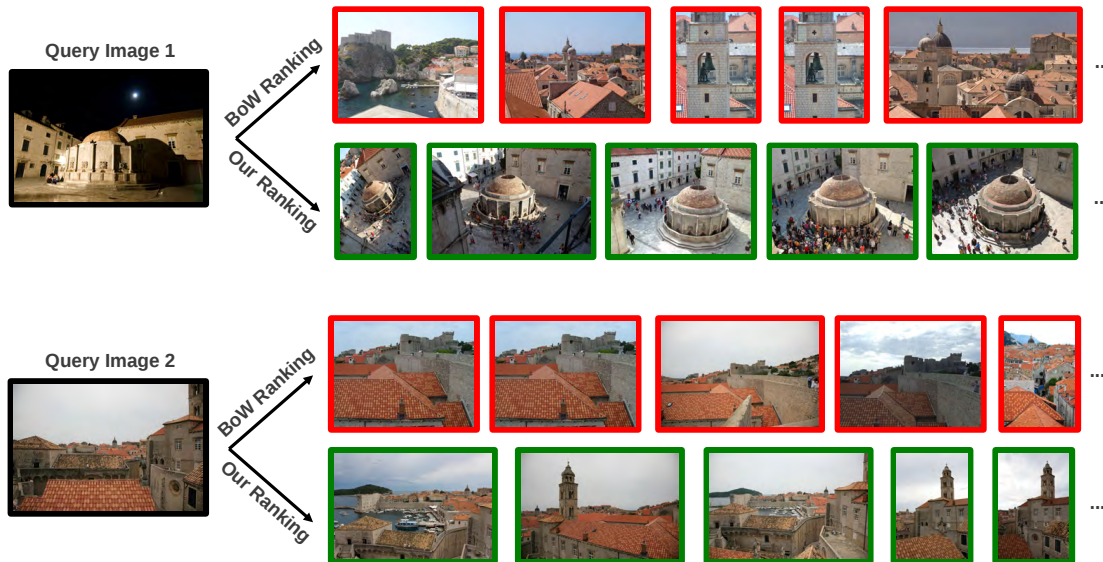


Figure 3.5: **Two example query images and their top 5 ranked results using our learned similarities and raw tf-idf BoW retrieval.** For each result, a green border indicates a correct match, and a red border indicates an incorrect match. The two example query images on the left are difficult for BoW retrieval techniques, due to drastically different lighting conditions (query image 1) and confusing features (rooftops in query image 2). However, with our discriminatively learned similarity functions, correctly matching images are ranked higher than with the baseline method.

Using this simple approach, we observe improvements in our ranked lists over raw BoW retrieval results, as shown in the examples in Figure 3.5. In particular, the top image in the ranked list is more often correct using our ranking based on learned similarities. However, when the top ranked cluster is incorrect, this method has the effect of saturating the top shortlist with similar images that are all wrong—there is a lack of *diversity* in the list, with the second-best cluster pushed further down the list. To avoid this, we propose several methods to encourage a diverse shortlist of images.

## 3.4 Diverse Shortlists

In this section, we first introduce a probabilistic method that uses the graph to introduce more diversity into the shortlist, increasing the likelihood of finding a correct match among the top few retrieved images. In addition, we demonstrate several techniques to introduce forms of regularization into our ranking to further improve recognition performance.

### 3.4.1 Probabilistic Reranking

In a way, our recognition problem is akin to the well-known Web search ranking problem, as compared to the standard formulation of the image retrieval problem. In our setting, rather than retrieve *all* instances relevant to a given query, it is more useful to retrieve a small set of results that are both *relevant* and *diverse* (see Figure 3.6 for an example), so as to cover multiple possible hypotheses—just as a Web search for the term “Michael Jordan” might productively return results for both the basketball player and the machine learning researcher.

In the information retrieval literature, a concept known as the *Probability Ranking Principle* captures the idea that ranking documents in decreasing order of probability of relevance provides an optimal ranking [43]. While this idea has merit if many documents are to be returned, if instead a retrieval system is to return a small number of documents, then returning the most probable set of documents may no longer be desirable. Instead, one can argue for *diversity* in such scenarios (as argued by Chen and Karger [8], among others in the information retrieval literature). In short, their argument is that in many cases a

better goal is to *maximize the probability of finding a relevant document among the top  $n$* . We extend this idea to our problem setting. However, unlike Chen and Karger, who must perform this probabilistic reasoning using document features alone, we have the benefit of having the graph structure as a form of supervision for performing such probabilistic reranking. Hence, as with learning similarity functions, we leverage our image graph once more.

The idea for our probabilistic approach for reranking the shortlist is, in some ways, a sort of “anti”-query expansion. In standard image retrieval, the idea of query expansion, or other forms of positive relevance feedback, is to use successful matches early in the shortlist to move other, similar images up the ranking, which typically increases recall. In our case, we use a form of blind *negative feedback* to increase the pool of diverse matches. In particular, in anticipation of the case where the first retrieved image is *not* a match to the query, we want to precompute the probability of the second image conditioned on this outcome, likely selecting an image dissimilar to this first match (and similarly for the third image conditioned on the first two being incorrect). How can we compute such conditional probabilities? This is where we turn to the image graph.

First, let us define our notation. Suppose we have a query image  $q$ . For a database image  $a$ , we define a random variable  $X_a$  representing the event that the query image matches image  $a$ :  $X_a = 1$  if image  $a$  is a match, and 0 otherwise. The calibrated probability values  $P_c(q)$  (which in what follows we denote as  $P_c$  as shorthand), computed using the algorithm in Section 3.3, give us an appearance-based estimate of these probabilities for each exemplar image  $c$ , i.e.,  $\Pr(X_c = 1) = P_c$ . Similarly, let us define  $\Pr(X_a = 1) = P_a$  for *any* database image  $a$ , using the simple heuristic above that the probability  $P_a$  of a non-exemplar

database image matching the query takes the maximum probability  $P_c$  of all neighborhoods that  $a$  belongs to.

To choose the top-ranked image for the query, we select the database image  $a$  with the highest appearance-based probability  $P_a$  (breaking ties by falling back to the BoW similarity, as in Section 3.3). However, to select the *second* ranked image  $b$ , we are instead more interested in the conditional probability  $\Pr(X_b = 1|X_a = 0)$  than its raw appearance-based probability  $\Pr(X_b = 1)$  alone. We denote this conditional probability as  $P'_b$ , which we can compute as:

$$\begin{aligned}
P'_b &= \Pr(X_b = 1|X_a = 0) = \frac{\Pr(X_b = 1, X_a = 0)}{\Pr(X_a = 0)} \\
&= \frac{\Pr(X_b = 1) - \Pr(X_b = 1, X_a = 1)}{1 - \Pr(X_a = 1)} \\
&= \frac{P_b - \Pr(X_b = 1|X_a = 1)\Pr(X_a = 1)}{1 - P_a} \\
&= \frac{P_b - P_{ba}P_a}{1 - P_a} = P_b \left( \frac{1 - \frac{P_{ba}}{P_b}P_a}{1 - P_a} \right)
\end{aligned} \tag{3.2}$$

where  $P_{ba}$  is shorthand for  $\Pr(X_b = 1|X_a = 1)$ , and denotes the conditional probability that image  $b$  matches the query given that image  $a$  matches the query. We can think of the last line in the derivation above as relating  $P'_b$  to  $P_b$  via an *update factor*,

$$\frac{1 - \frac{P_{ba}}{P_b}P_a}{1 - P_a} \tag{3.3}$$

that depends on  $P_a$  (the probability that the top ranked image  $a$  matches the query) and  $P_{ba}$  (the conditional probability that  $b$  matches given that  $a$  does). How do we compute  $P_{ba}$ ? This is where the image graph comes in. The intuition here is that the more similar  $b$  is to  $a$ —i.e., stronger the connection between  $a$  and  $b$  in the image graph—the higher  $P_{ba}$  should be. In particular, we define

$$P_{ba} = \frac{N(a, b)}{N(a)}, \tag{3.4}$$

i.e., the number of shared features between  $a$  and  $b$  divided by the total number of matchable feature points in  $a$ . We use this empirical measurement of the percentage of shared features out of all  $a$ 's matchable features to estimate  $P_{ba}$ . Note that in general  $P_{ab} \neq P_{ba}$ , i.e., this similarity measure is asymmetric (as expected for conditional probabilities). These graph-based similarity measures are pre-computed along with the Jaccard indices  $J(a, b)$  described in Section 3.2.

The update factor in Eq. (3.2) has an intuitive interpretation: roughly speaking, if image  $b$  is very similar to image  $a$  according to the graph (i.e.,  $P_{ba}$  is large relative to  $P_b$ ), then its probability score is downweighted (because if  $a$  is an incorrect match, then  $b$  is also likely incorrect). On the other hand, if  $b$  is not connected to  $a$ , its score will tend to be slightly boosted by this update factor, as the fact that it is dissimilar to  $a$  is at least weak evidence in support of  $b$  if  $a$  does not match the query. More precisely, the ratio  $\frac{P_{ba}}{P_b}$  in the update factor depends on whether image  $b$  is more similar to image  $a$  or to the query image  $q$ , where “similarity to  $a$ ” is measured using the image graph, and “similarity to  $q$ ” is measured using the calibrated SVM outputs.

In practice, we do not want to apply this update too quickly, for fear of downweighting many images based on the evidence of a single mismatch. To regulate this factor, we introduce a parameter  $\alpha$ , and define a regularized update factor:

$$\frac{1 - \alpha \frac{P_{ba}}{P_b} P_a}{1 - \alpha P_a}. \quad (3.5)$$

If  $\alpha = 0$ , the update has no influence on the ranking result, and if  $\alpha = 1$ , it has its full effect. We use  $\alpha = 0.9$  in our experiments.

Next, we select the third image, now based on the conditional probability that the first two images fail to match, and so on for the fourth image, etc.

For selecting the third image  $c$ , we are interested in the conditional probability  $P(X_c = 1|X_b = 0, X_a = 0)$ . We make two simplifying assumptions in computing this conditional probability. First, we assume that  $X_b = 0$  and  $X_a = 0$  are independent events. Although we compute  $b$  by maximizing a conditional probability on  $a$ , by construction  $b$  will very often not be closely connected to  $a$  in the graph; hence this conditional independence is a reasonable assumption. Second, we assume that the event “both  $c$  and  $b$  match query” is independent of  $a$ ’s failure to match the query. Intuitively, this assumes that the overlapping area between image  $c$  and  $b$  doesn’t intersect  $a$ , i.e.  $b$  and  $a$ ’s intersections with  $c$  do not overlap. Again, this assumption is likely to hold because  $b$  is chosen to be distinct from  $a$  in the first place by the diversity algorithm. Hence, we have

$$\Pr(X_c = 1, X_b = 1|X_a = 0) = \Pr(X_c = 1, X_b = 1). \quad (3.6)$$

Given these assumptions, the update factors at each round become very simple to compute: after selecting an image, we simply replace each probability value  $P_i$  for each remaining image with the updated version  $P'_i$ , and treat this as factoring in all previous updates. We then compute the next image via the same update factor in Eq. (3.5), but using these new probabilities. In other words, at each round, the evidence from all previously ranked images are integrated in an iterative fashion by repeatedly updating all candidate images’ probability scores using the update factor in Eq. (3.5). A more complete derivation for this method can be found in the Appendix. At each iteration, we select the image for the shortlist that maximizes the modified probability at that iteration. One way to look at this method is as another kind of greedy algorithm for covering the graph, but one in which we also take into account appearance similarity to the query image.



Figure 3.6: **An example query image and the top 5 ranking results using our method with and without probabilistic ranking.** Green borders indicate correct matches, and red borders incorrect ones. Without probabilistic ranking, our algorithm generates a top 5 set of results that are similar, and all incorrect. With probabilistic reranking, more diversity is encouraged in the top ranking results, leading to several correct images among the top 5 results.

### 3.4.2 Bag-of-Words Regularization

Another issue we have found to be important is that while our learned discriminative models generally perform well, for certain rare query images, our models consistently perform poorly. This is perhaps due to sparser parts of the graph having relatively few training examples, as is evident in the sparser parts of Figure 3.2. For this reason, we found it helpful to use the original tf-idf-based similarities as a way of “regularizing” our rankings, in case of query images for which our models perform poorly. We have explored three ways of doing this:

- **Fall-back strategy.** First, for query images where *all* models give a probability score below a minimum threshold  $P_{\min}$  (0.1 in our tests), we fall back to tf-idf scores, as we found low probability scores unreliable for ranking. (In our experiments, this case occurs in  $\sim 5\%$  of queries.)



- **Averaging of scores.** Second, to regularize our probability scores in case of overfitting in the learning process, we take a weighted average of our probability scores and a tf-idf-based probability value; this value is given by a logistic regressor fitted using matching and non-matching image pairs in the image graph.
- **Interleaving.** Third, we found that our learned models and the original tf-idf scores sometimes performed in a complementary way; while our models work well for many queries, some query images still performed better under tf-idf. Thus, as a way of introducing more diversity, and an alternative for the fall-back strategy, we *interleave* the results of the two rankings. The order of interleaving is determined by the maximum value of our probability outputs, which we use to determine the confidence of our original ranking. If this value is less than a threshold (we use 0.1), then BoW ranking goes first, and vice versa.

In our experiments, we use the simple fall-back strategy by default in all of our experiments, and additionally evaluate a combination of averaging and interleaving as a stronger form of tf-idf regularization. Figure 3.7 illustrates how we combine the tf-idf scores and our own learned probabilities to form a (dynamic) ranking of the database images.

**Discussion.** One could argue that the necessity of increasing diversity in our case is caused by the decision in Step 3 in Section 3.3 to generate a ranked list of the entire image set rather than only the exemplar images. For instance, perhaps choosing the single image with the highest BoW similarity to the query image from each neighborhood might solve the problem. We have tried this approach (**GBP+MaxBoW** in Table 4.4) on the Dubrovnik dataset, and found that overall

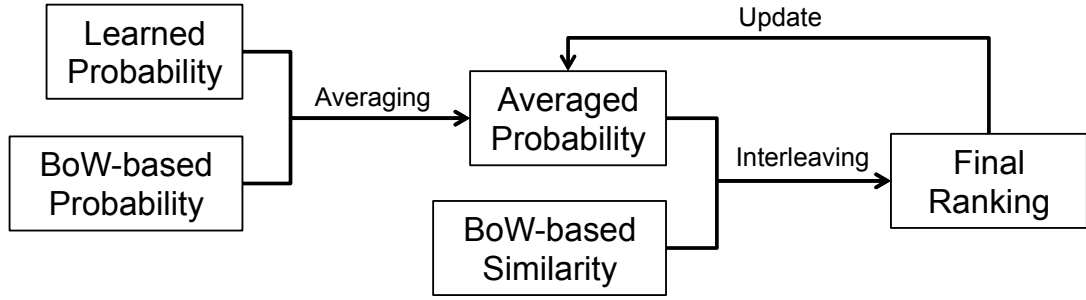


Figure 3.7: **The overall procedure for our BoW-based regularization of the shortlists.** In each round of verification for the query image, we select the best scoring image (alternating between averaged probability and BoW similarity) in the **Final Ranking** step. If this selected image doesn't match the query, we update the average probabilities for the remaining images using Eq. (3.5).

the diversity approach performed better. It is unclear how each neighborhood should be sampled, either by BoW similarity, or other methods, that ensures not only that query image has a high probability to be successfully matched if the query image is in that neighborhood, but also the true neighborhood doesn't appear too far down the ranking list if the query image is not.

### 3.5 Evaluation

In this section, we evaluate different variants of our proposed method, and compare to several baseline approaches, on a range of datasets.

As discussed in Section 3.3, a key bottleneck of image retrieval-based location recognition systems is the quality of the image ranking—we want the first true match to a query image to rank as high in the list as possible, so we have to run the (more expensive) detailed matching and geometric verification procedure on as few images as possible. Hence, to evaluate an ordering of images in the shortlist, we use accuracy at top  $k$  ( $k \in \{1, 2, 5, 10\}$ ), i.e., the percentage of query images for which at least one correct match exists in the top  $k$  retrieved results. Note that all the methods we test are compared on an equal footing, based purely on the shortlist they generate, without use of RANSAC-based verification before examining results. In all cases we apply detailed verification on each short-listed image sequentially until the first true match is found, at which point a localization is achieved.

#### 3.5.1 Datasets and Preprocessing

We evaluated our algorithm on several location recognition datasets, including the Dubrovnik and Rome datasets [29], the Aachen dataset [46], and a much larger Landmarks dataset [30] consisting of 1,000 separate landmarks across the globe; these datasets are summarized in Table 4.1, along with statistics over the neighborhoods we compute for each dataset.

To represent images as BoW histograms, we learned two kinds of visual vo-

Dataset	# Queries	# DB Images	# Clusters	Avg. Cluster Size
Dubrovnik [29]	800	6,044	188	103.9 ( $\pm 103.4$ )
Rome [29]	1,000	15,179	352	146.5 ( $\pm 180.8$ )
Aachen [46]	369	4,479	161	41.2 ( $\pm 37.8$ )
Landmarks [30]	10,000	206,162	8,803	67.3 ( $\pm 102.7$ )

Table 3.1: **Summary of datasets used in our experiments, along with their computed neighborhoods.** Each row summarizes a dataset, showing the number of query images in the test set, the total number of images in the database, and the number of neighborhoods our algorithm selects in order to cover each image graph. The representative neighborhoods (clusters) are found using graphs whose edge weights are defined using the Jaccard index, and thresholded using a value of 0.01. The right-most column shows the average cluster size in each dataset, along with the standard deviation in the cluster sizes.

cabularies [38]: one vocabulary learned from each dataset itself (a **specific** vocabulary) and another shared vocabulary learned from  $\sim 20,000$  randomly sampled images from an unrelated dataset (a **generic** vocabulary). Each vocabulary is of size 1M. As our ground truth, we count an image pair as matching if the pair has at least 12 inlier feature matches after pairwise matching and geometric verification.

### 3.5.2 Evaluation Methodology

For each dataset, we compare several algorithms for recognizing each query image:

- (a) **BoW**: Standard tf-idf weighted bag-of-words (BoW) image retrieval [38].
- (b) **BoW+RR**: A probabilistic reranked version of (a) using our method described in Section 3.4.1, and the calibrated BoW similarities as probabilities. To calibrate, we randomly select equal number of matching and non-

matching image pairs from the database and fit a logistic regression model using the BoW similarities of these pairs as the input feature. This allows us to evaluate reranking independent of our learning method.

- (c) **GBP**: (“Graph-based probability.”) Our graph neighborhood based similarity learning technique (Section 3.3).
- (d) **GBP+RR**: Our learning method using probabilistic reranking (Section 3.4.1).
- (e) **GBP+RR+BoW**: Our method with both probabilistic reranking and the BoW regularization (Section 3.4.2).

In addition, for one dataset (Dubrovnik, with a specific vocabulary), we also compare to a range of other baselines, including a more recent retrieval method using co-occurring sets of visual words [10] and a baseline that uses geotags on images as a supervisory signal, inspired by work on learning confusing features [47, 25]. In particular, for this last baseline, we randomly select a set of exemplar images, define the nearest neighbors using GPS positions (we use 200 nearest neighbors in our experiments) as positives and the remaining images as negatives and then use the same learning and retrieval techniques described above using these neighborhoods. In addition, on the Dubrovnik dataset, we evaluate two alternative learning approaches: a global distance metric learned using pairs of matching and non-matching image pairs in the graph [5], and our technique but trained using *every* database image as an exemplar (i.e., learning a per-image distance metric). Finally, also on the Dubrovnik dataset, we evaluate an alternative approach to combine BoW similarity and our GBP score (as mentioned in the discussion towards the end of Section 3.4.2), i.e. choose one image from each neighborhood with the maximum BoW similarity using the order

determined by GBP score.

Note that while we use a relatively simple image representation (weighted BoW histograms), our method is orthogonal to many other improvements to bag-of-words models [2], and can generalize to more sophisticated feature representations.

**Experimental details.** We construct a Jaccard-index weighted image graph  $\mathcal{G}$ , and threshold by  $\tau = 0.01$  to obtain a set of positive image pairs. From the graph  $\mathcal{G}$ , we choose exemplar images (neighborhoods) and learn  $L_1$ -regularized linear SVMs and logistic functions as described in Section 3.3. We found that  $L_1$  regularized SVMs perform slightly better than  $L_2$  regularized SVMs for our problem, and the  $L_1$  models also have the advantage of yielding much sparser weight vectors (typically  $< 1\%$  non-zeros compared to typically  $> 99\%$  non-zeros for  $L_2$  regularized models) and hence faster query times. For each cluster, we use all the available positive examples (i.e., cluster sizes in Table 4.1), and sample roughly 5 times as many negative examples as positives. For each learning problem, one-third of the training data is held out at random for validation, and all training data is used for logistic regressor training. We use *liblinear* [14] for SVM and logistic regressor training. For each query image, we compute the estimated probability of it matching each cluster, and obtain the initial ranking of the database images as described in Section 3.3. We show the results of our method (a) ranking with just the graph-based probability scores (**GBP**), (b) ranking neighborhoods using graph-based scores then choose the image in each one of them with the maximum BoW similarity in the ranked order (**GBP+MaxBoW**), (c) reranking using our diversity measure (**GBP+RR**), and (d) the stronger form of BoW regularization (**GBP + RR + BoW**) using a weighted

average of the two probability scores, with a weight of 5/6 on our GBP score, and 1/6 on the tf-idf-based probability score, as well as an interleaving ranking (described in Section 3.4).

### 3.5.3 Runtime analysis

Our approach involves additional overhead at training time, as well as a small amount of extra work at runtime. Suppose we have an image set of size  $S$ , and each image is encoded in an  $n$ -dimensional BoW vector.  $\sigma$  is the average ratio of non-zero elements in the BoW vectors, and  $m$  is the number of neighborhoods selected by our algorithm. Our greedy algorithm for selecting neighborhoods takes time is  $\mathcal{O}(mS)$ , as in each iteration we consider at most  $S$  candidate images. Training the linear SVMs tanks time  $\mathcal{O}(mSn\sigma)$ , since the total number of training examples (both positives and negatives) is  $\mathcal{O}(S)$ . Note that SVM training is also faster with stronger regularization.

At runtime, for a query, we first need to compare the BoW vector of a query image to the  $m$  learned weight vectors, then compare it against all images for BoW regularization, hence the running time for computing these similarities is  $\mathcal{O}((m + S)n\sigma)$ . To generate the diverse shortlists of length  $k$ , each time when we choose to place an image on the ranking list, we need to recompute probabilities from  $S$  candidates and find the maximum one. Hence, the running time of these operations is  $\mathcal{O}(kS)$ . In summary, to generate a length- $k$  shortlist for a query image, the total running time is  $\mathcal{O}((m + S)n\sigma + kS)$ .

To put these variables into perspective in practice, for Dubrovnik dataset, there are  $S = 6044$  images and  $m = 188$  neighborhoods (Table 4.1), and the

Dubrovnik (Specific Vocab.)					
Method	top1	top2	top5	top10	mAP
BoW [51]	87.50%	92.75%	97.62%	98.50%	0.401
BoW+RR	87.50%	93.38%	96.63%	97.50%	0.058
Co-ocset [10]	87.50%	92.50%	97.50%	98.62%	0.389
GPS Model	87.87%	89.75%	91.75%	93.25%	0.367
Global Model [5]	85.37%	91.63%	95.87%	97.38%	<b>0.643</b>
Instance Model	90.00%	95.13%	98.12%	98.50%	<b>0.643</b>
GBP	<b>94.38%</b>	96.37%	98.25%	98.50%	0.626
GBP+MaxBoW	<b>94.38%</b>	<b>97.50%</b>	99.00%	99.13%	0.200
GBP+RR	<b>94.38%</b>	96.25%	98.62%	99.13%	0.273
GBP+RR+BoW	94.25%	97.12%	<b>99.37%</b>	<b>99.50%</b>	0.122
Dubrovnik (Generic Vocab.)					
Method	top1	top2	top5	top10	mAP
BoW	75.88%	83.00%	90.88%	95.63%	<b>0.512</b>
BoW+RR	75.88%	83.62%	93.25%	<b>96.25%</b>	0.065
GBP	81.25%	85.13%	88.13%	90.00%	<b>0.512</b>
GBP+RR	81.25%	83.87%	89.88%	95.13%	0.151
GBP+RR+BoW	<b>81.88%</b>	<b>90.00%</b>	<b>94.00%</b>	96.00%	0.085

Table 3.2: **Recognition performance on Dubrovnik dataset.** The abbreviations for each method are defined in Section 3.5.2.

average non-zero ratio for BoW vectors is  $\sigma = 2.5\%$ . To give some examples measured training and querying running times of our approach, the average training time (SVM training and calibration) is about 40 seconds per model, and the average query time is about 5 seconds per query for  $k = 10$ . In addition, since we need to store an additional weight vector for each neighborhood, the memory usage of our algorithm has the additional overhead of storing  $m$  weight vectors (of size  $\mathcal{O}(mn\sigma)$ ) compared to standard image retrieval methods.

### 3.5.4 Results

The results of all experiments are shown in Table 3.2 and Table 3.3. We show results using both specific and generic vocabularies for the Dubrovnik dataset



Rome					
Method	top1	top2	top5	top10	mAP
BoW	97.40%	98.50%	99.50%	99.60%	0.674
BoW+RR	97.40%	98.70%	99.10%	99.10%	0.047
GBP	97.80%	98.70%	99.30%	99.30%	<b>0.789</b>
GBP+RR	97.80%	98.80%	99.30%	<b>99.70%</b>	0.403
GBP+RR+BoW	<b>97.90%</b>	<b>99.00%</b>	<b>99.70%</b>	<b>99.70%</b>	0.259

Aachen					
Method	top1	top2	top5	top10	mAP
BoW	80.76%	83.47%	86.45%	88.35%	0.431
BoW+RR	80.76%	82.66%	86.45%	88.62%	0.069
GBP	<b>82.38%</b>	84.55%	86.72%	88.35%	<b>0.459</b>
GBP+RR	<b>82.38%</b>	83.74%	87.26%	88.89%	0.205
GBP+RR+BoW	<b>82.38%</b>	<b>84.82%</b>	<b>88.08%</b>	<b>89.16%</b>	0.185

Landmarks					
Method	top1	top2	top5	top10	mAP
BoW	58.42%	64.10%	71.22%	75.81%	0.115
BoW+RR	58.42%	65.97%	77.04%	80.76%	0.064
GBP	62.58%	64.59%	66.90%	68.35%	<b>0.471</b>
GBP+RR	62.58%	65.19%	75.34%	<b>81.17%</b>	0.084
GBP+RR+BoW	<b>64.60%</b>	<b>73.03%</b>	<b>78.01%</b>	81.01%	0.052

Table 3.3: **Recognition performance on Rome, Aachen and Landmarks datasets.** The abbreviations for each method are defined in Section 3.5.2.

in Table 3.2, and only using the specific vocabulary for the other datasets in Table 3.3, as we found that across the datasets the specific vocabulary outperformed the generic vocabularies. (We were interested in seeing whether our learning methods could overcome the disadvantages of the generic vocabulary, but found that there was still a significant advantage to specific vocabularies even with learning.)

**Graph-based methods.** A few trends are evident from the results. Our graph-based probability method (**GBP**), by itself, consistently improve results for the top1 and top2 rankings over the baseline BoW method; the improvement in top1 success rate ranges from a small amount for the Rome dataset, to nearly  $> 7\%$  for

the Dubrovnik dataset (with specific vocabulary); the average improvement is 3.8% across all datasets. However, the performance of the GBP method increases more slowly than the baseline tf-idf ranking as a function of  $k$ , and for the top5 and top10 rankings GBP performs worse in many cases. However, once we reintroduce diversity through probabilistic reranking (**RR**), our results improve on average for these longer shortlists (3.9% on average across our datasets for top10). We observe small additional gains when regularizing our learned results with the tf-idf BoW scores. Figure 3.8 shows the top- $k$  accuracy performance on the Dubrovnik dataset (trained with a generic vocabulary) as a function of  $k$ . Although the GBP performance alone is worse than our BoW baseline for larger  $k$ , probabilistic reranking and BoW regularization improve the performance to be better than the baselines.

Note that there is a significant variety in the structure and difficulty of our datasets. Rome is a relatively easy dataset, consisting of many separate scenes (most of which are small and compact), while Dubrovnik has a more interesting graph structure, consisting of a single large connected component spanning many viewpoints across a city. Landmarks consists of the union of many separate 3D models, some large, some small, and is the most difficult dataset due to its large size. For the Landmarks dataset, the improvements we see using our method are particularly large, over 5.36% for top10 over the bag-of-words baseline, with the diversity reranking being the most significant factor in this improvement. This suggests that our method may be particularly effective for improving recognition performance with large datasets. We found that adding diversity to the raw BoW method (i.e., **BoW+RR**) did not consistently perform better than the standard BoW method. However, for the Landmarks dataset diversity significantly improves the BoW method (from 76% to 81% for top10) just

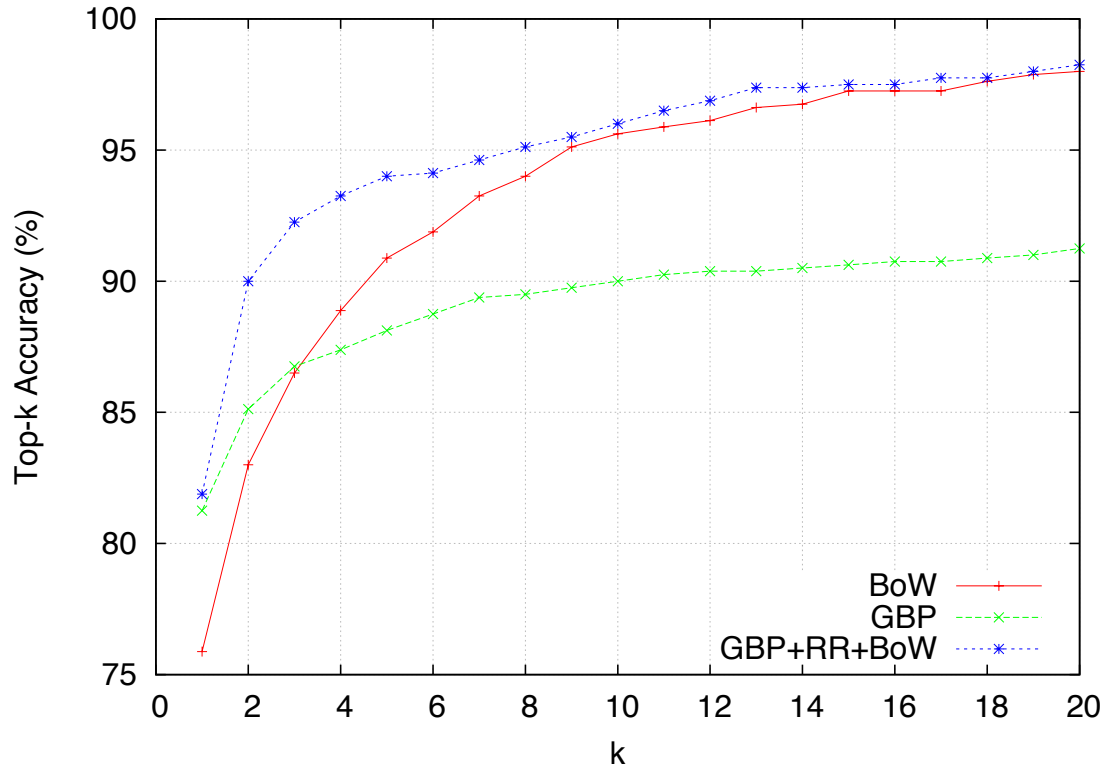


Figure 3.8: **Top-k accuracy on the Dubrovnik dataset (generic vocabulary).**

as it does our GBP method.

**Other baseline methods.** In the Dubrovnik (Specific Vocab.) results, our full method outperformed each of the unsupervised baseline methods (BoW, BoW+RR and Co-ocset [10]). As described above, we also evaluated several supervised baselines. The GPS-based model (**GPS Model**) did not improve over the unsupervised baselines (in fact, did worse in general). This is likely due to the less careful choices of training examples compared to those based on image graph. The globally trained similarity metric (**Global Model**) also performed worse in general compared to the unsupervised methods, at least as measured by how often a correct result is in the top- $k$  matches. Interest-

ingly, however, it does significantly improve the mAP (mean average precision) score (computed on a ranking of all images), suggesting that this method is much better at globally ranking the images than it is at our recognition task—in other words, it is doing better at the problem of retrieving all relevant instances. The per-image classifiers (**Instance Model**) perform best among the baselines, but the GBP method still performs better. We believe this is due to the nature of image graphs for unstructured collections, where some nodes have many neighbors, and others (e.g. very zoomed-in images) have only a few; training and calibration for these low-degree nodes can result in models that overfit the data and contaminate the global ranking. In addition, increasing the diversity (+RR) and strong regularization using BoW results (+BoW) both are beneficial in improving our original ranking results (though these techniques result in a smaller mAP score; this again suggests an interesting tradeoff between retrieval and recognition performance). Compared to our proposed BoW regularization method (+BoW), choosing the image with the maximum BoW similarity in the ranked order by GBP scores (+MaxBoW) performs better for top1 and top2 images, but worse for top5 and top10.

For both Dubrovnik and Rome, our top 10 success rate (99.5% on Dubrovnik and 99.7% on Rome) is comparable to the results of [30] (100% / 99.7%), which uses direct 3D matching and requires much more memory and expensive nearest neighbor computations. This effort is needed since these methods store entire 3D point clouds (often with millions of individual points), as well as a 128-byte SIFT feature for each point—often requiring many gigabytes of memory. In our case, we only store a set of quantized features, BoW image vectors, and sparse weight vectors. For example, for the Dubrovnik dataset, our unoptimized Python implementation of our system uses about half the memory as

the C++ implementation of [30]. Our performance on Aachen dataset (89.16%) also rivals that of [46], where their best result 89.97% is achieved with a relatively expensive method, while we only use the compact set of weights learned from neighborhoods. In all cases, we improve the top  $k$  accuracies over BoW retrieval techniques, resulting in a better ranking for the final step of geometric consistency check procedure.

## CHAPTER 4

### MINIMAL SCENE DESCRIPTIONS FROM STRUCTURE FROM MOTION MODELS

In Chapter 3, we explored using the image graph to improve image retrieval based location recognition techniques — given one answer to the question “what is a place”. In this chapter, we approach this question from a different angle, using another kind of graph representing visibility between images and 3D points.

Indeed, there has been a line of recent work that utilizes structure-from-motion techniques to construct large-scale databases of images and 3D point clouds [52, 1], for a variety of applications, including location recognition [58, 21, 29, 44, 30]. These location recognition methods often directly match features (such as SIFT [31]) in a query image to descriptors associated with 3D points. These databases of 3D points, however, can be very large—ranging in size from a few million points in a single location, to hundreds of millions when multiple places are considered together [30]. For purposes of modeling and visualization, the denser the 3D points the better. However, for other applications, such as recognition, there are advantages in having fewer points, such as reduced memory and computation requirements. Therefore, in this Chapter, we are interested in an interesting question: how much data do we need to describe a location? What is a minimal description of a place?

One way to make this question concrete is to define it as a visibility covering problem [29, 37]: every possible image that one could take of the location should see some minimal number of 3D points stored in the reconstruction. Such a covering constraint makes it likely that a new image of the scene will match a

sufficient number of 3D points to enable pose estimation. Based on this idea, prior methods have used the visibility relationships between images and points in the database to compute reduced 3D point sets that cover the database images. However, another important factor is **distinctiveness**: in order to ensure accurate matching, one should select a subset of points that are distinct (rather than selecting points with very similar appearance). In this chapter, we show that by computing a reduced scene description that takes into account both coverage and distinctiveness, one can compute very compact models that maintain good recognition performance.

We incorporate these considerations into a new point selection algorithm that predicts how well new images will be recognized using a probabilistic approach. We evaluate our algorithm on several standard location recognition benchmarks, and show that our computed scene representations consistently yield higher recognition performance compared to previous model reduction techniques.

## 4.1 Related Work

Our algorithm is inspired by the  $K$ -covering algorithms used in prior 2D-to-3D matching-and-pose-estimation systems [29], but has two key differences compared to this prior work. First, we consider **point appearance** in order to select visually distinctive points; second, our selection algorithm uses a **probabilistic model** of visibility as opposed to the strictly combinatorial methods used in prior work.

Park et al. select a subset of 3D points using mixed-integer quadratic programming [37]. A limitation of this approach is the computational hardness of the underlying optimization problem, making it difficult to scale up to large, world-wide datasets [30]. Choudhary et al. model point visibility probabilities to guide a 3D matching and pose estimation algorithm at runtime [9]. Our goal is different in that we aim to select a subset of database points in advance without knowledge of a specific query image. Although the probabilistic formulation in [9] works well for modeling inter-point and inter-image relationships given a query image, a direct adaptation of this approach, by modeling inter-point relationships, is computationally prohibitive due to the non-linear composition of probabilities and combinatorial explosion of point sets. Irschara et al. use synthesized views from a point cloud to cover a 3D scene [21]. We take a different approach that directly models image-point relationships, but it is possible to extend our method by adding synthesized views as an additional set of “images” to cover in our algorithm.

Our use of distinctiveness as a factor in selecting points is related to prior work on identifying “confusing” features for recognition [47, 25]. For instance,



Knopp et al. use image geotags to identify features that appear at multiple disparate locations, and incorporate this information into a bag-of-words recognition framework [25]. In our case, we select individual 3D points rather than visual words, and do not require GPS information. Hence, we have more fine-grained control over the set of features to avoid or select. Philbin et al. use descriptor learning to find a non-linear transformation of the descriptor space in order to better separate true feature matches from false ones [40]. Our work is orthogonal in that it seeks to find a subset of distinctive features in the standard descriptor space, but our algorithm could easily incorporate learned distance metrics.

Finally, our method is related to prior work on feature selection for identifying discriminative features [55]. Li and Kosecka propose a method for identifying highly discriminative individual features for a discrete set of locations [26]. Doersch et al. propose to use a discriminative clustering approach to find visual elements that are most distinctive for a city (such as Paris) [13]. Cao et al. use discriminative learning on clusters of images to define different distance metrics representing locations [6]. Turcot and Lowe use image matching process to select reliable visual words [54]. While we also favor distinctive image descriptors in our approach, we again do so at a much more fine-grained level than bag-of-words models, and also seek to maximize *coverage* of the dataset as well as discriminability.

## 4.2 Computing Minimal Point Sets

In this section we describe our algorithm for computing minimal scene representations, starting with background.

We begin by running structure from motion (SfM) to reconstruct one or more scenes that form a database for use in recognizing and posing new images [1]. The result of running SfM on an image set  $\mathcal{I}$  of size  $m$  is a 3D point set  $\mathcal{P}$  of size  $n$ , (typically  $n \gg m$ ), as well as a *visibility matrix*  $M$  of size  $m \times n$  defining the visibility relationships between images and points, where  $M_{ij} = 1$  if point  $P_j$  is visible in image  $I_i$  in the reconstructed 3D model, and  $M_{ij} = 0$  otherwise. This matrix can also be interpreted as a bipartite *visibility graph*  $\mathcal{G}$  on images and points, where an edge links each image to each point visible to that image. In addition to a 3D location, each point  $P_j \in \mathcal{P}$  also has a feature descriptor, for instance the average SIFT descriptor of the features used to triangulate  $P_j$  [29]. Such a reconstruction can be used to recognize the pose of new photos via 2D-to-3D matching and pose estimation techniques [45, 30]. To register a query photo, we extract features from the photo, match them to features in the database using approximate nearest neighbors [3], and robustly estimate the absolute camera pose using the matched points. Key to this process is to find a sufficient number of correct feature matches between the query image and the database.

Our goal is to compute a more compact database with a much smaller set of points  $\mathcal{P}' \subset \mathcal{P}$ , such that  $\mathcal{P}'$  captures as much of the information in the full model as possible. In particular, we wish to be able to correctly register as many new query images to the subset  $\mathcal{P}'$  as possible.

**$K$ -cover algorithm.** The prior work of Li et al. [29] begins with the assumption

that the distribution of query images is similar to the distribution of database images. Hence, they use coverage of the database images as a proxy for coverage of query images of interest. They formulate this as a *K-cover (KC) problem* on the visibility graph  $\mathcal{G}$ : select a minimum subset of points such that each database image sees at least  $K$  points in the subset. Finding such a minimum set is a combinatorially hard problem, and so they use a greedy algorithm that starts with the empty set, and incrementally adds the next point  $P_j$  that maximizes the *gain* in coverage achieved by adding  $P_j$  to the current set  $\mathcal{P}'$ :

$$G_{KC}(j, \mathcal{P}') = \sum_{I_i \in \mathcal{I} \setminus C} M_{ij} \quad (4.1)$$

where  $C$  is the set of images that are already “covered” at least  $K$  times by points in  $\mathcal{P}'$ , i.e.  $C = \{I_i | \sum_{P_t \in \mathcal{P}'} M_{it} \geq K\}$ , and hence do not contribute to the gain of a point. This algorithm runs until no further point contributes a positive gain (or until a target percentage of images, e.g. 99%, are covered). We denote the gain function in Eq. 4.1 as  $G_{KC}$  in reference to the  $K$ -cover algorithm that it corresponds to. In short, the  $K$ -cover algorithm ensures good coverage of the database images by greedily selecting points that “cover” the most uncovered images. In what follows, we will define alternative gain functions that model additional aspects of the recognition problem.

In order to compute the pose of a query image, a minimum of three feature matches to 3D points are required (or four, in case the camera intrinsics are unknown, or six, in case a full projection matrix is desired). In practice, many more matches are desirable; hence, Li et al. use large values for  $K$  (e.g.,  $K = 100$ ).

**Our approach.** Like the  $K$ -cover algorithm, we use a greedy algorithm to incrementally create a compact subset of points. However, we define a different gain

function using a probabilistic framework: we seek to select a subset of points that maximizes the *probability* of registering a new image, while minimizing the number of points selected. At a high level, our approach considers two aspects:

1. **coverage**, i.e., the subset covers the scene in that any new image has a high probability of seeing a large number of points, and
2. **distinctiveness**, i.e., the features we select are sufficiently distinct from one another in appearance.

We found that both of these aspects contribute to the goal of finding good feature matches between a query image and the database.

#### 4.2.1 Maximizing expected coverage

We first describe our approach to ensuring coverage. As in the  $K$ -cover algorithm, we want to select points in such a way that any query image that matches the scene sees a certain minimum number of points. The  $K$ -cover algorithm treats covering the database in a strictly combinatorial way. However, because feature detection and matching are noisy processes, we instead view this problem from a probabilistic perspective, where the database images are treated as samples from some underlying distribution of images of a scene. Hence, we consider points being visible in images as random events with certain probabilities, rather than as simply binary variables. In particular, we define that each point  $P_j$  is visible in each database image  $I_i$  with probability  $p_{ij}$ . We experimented with different methods for defining these probabilities, such as forms of smoothing the bipartite visibility matrix  $M$ . We found that simply using a

constant value  $p$  for  $M_{ij} = 1$  and 0 for  $M_{ij} = 0$  worked well, though we note that finding better ways to estimate  $p_{ij}$  is an interesting avenue for future work.

Given this probabilistic model of point visibility, our goal is to find a subset of points  $\mathcal{P}'$  that maximizes the probabilities of each image seeing at least  $K$  points in  $\mathcal{P}'$ . More formally, let  $v_{i,\mathcal{P}'}$  denote the random variable representing the number of points in the selected set  $\mathcal{P}'$  that are visible in database image  $I_i$ . Our objective is to maximize

$$\sum_{i \in \mathcal{I}} \Pr(v_{i,\mathcal{P}'} \geq K) \quad (4.2)$$

i.e., the sum of probabilities that each image  $I_i$  sees at least  $K$  points in the selected set  $\mathcal{P}'$ . If we assume that each observation of a point is independent, then the distribution of each random variable  $v_{i,\mathcal{P}'}$  is described by a binomial distribution if  $p_{ij}$  is a constant  $p$ , or a Poisson binomial distribution (a generalization of the binomial distribution) if  $p_{ij}$  varies for each point observation.

To balance the objective in Eq. 4.2 with the desire for a compact model, we set a target probability  $p_{\min}$ , and seek that  $\Pr(v_{i,\mathcal{P}'} \geq K) \geq p_{\min}$  hold for all images  $I_i$ , with  $|\mathcal{P}'|$  as small as possible. To achieve this goal, we adopt the greedy approach of repeatedly selecting the point  $P_j$  that maximizes the following gain function:

$$G_{KCP}(j, \mathcal{P}') = \sum_{i \in \mathcal{I} \setminus C} \Pr(v_{i,\mathcal{P}' \cup \{P_j\}} \geq K) - \Pr(v_{i,\mathcal{P}'} \geq K) \quad (4.3)$$

Here, KCP refers to our probabilistic  $K$ -cover algorithm. This measures the gain in *expected coverage* of the database achieved by adding a point  $P_j$  to the selected set. As in Eq. 4.1,  $C$  is the set of already-covered images, but in a probabilistic sense:  $C = \{I_i | \Pr(v_{i,\mathcal{P}'} \geq K) \geq p_{\min}\}$ . In other words, our algorithm starts with  $\mathcal{P}' = \emptyset$ , and repeatedly adds the point  $P_j$  that maximizes the expected gain

defined in Eq. 4.3. We describe this algorithm in more detail in Section 4.2.3.

However, there is a bootstrapping problem with this formulation. If, at some point in the algorithm (e.g., at the beginning), an image  $I_i$  sees fewer than  $K - 1$  points in  $\mathcal{P}'$  (i.e.,  $\sum_{j \in \mathcal{P}'} M_{ij} < K - 1$ ), then the gain for adding any new point to  $\mathcal{P}'$  w.r.t. image  $I_i$  is zero. In this case, we cannot effectively compute the gain in coverage of image  $I_i$  by adding another point. To avoid this problem, we bootstrap by first choosing an initial set of points that “ $K$ -covers” the images, i.e. a set  $\mathcal{P}'$  that satisfies  $\sum_{P_j \in \mathcal{P}'} M_{ij} \geq K$  for each image  $I_i$ . We next describe how we select an initial set of distinctive points in Section 4.2.2, and then show how to select a final covering set by maximizing Eq. 4.3.

## 4.2.2 Appearance-aware initial point set selection

We now describe how we select the initial point set that “ $K$ -covers” the images while considering distinctiveness of appearance. Consider a particular point  $P_j$ .  $P_j$  is associated with a set of individual SIFT descriptors  $\mathcal{D}_j$  in two or more database images; in our work, for compactness, we represent  $P_j$  with the centroid of these descriptors,  $\bar{D}_j$ . Assuming that the database descriptors  $\mathcal{D}_j$  are representative of other query descriptors that will later match this point, we want these descriptors  $D \in \mathcal{D}_j$  to be closer to  $\bar{D}_j$  in SIFT space than to the descriptor of any other selected point. To motivate this approach, consider Figure 4.1, which shows two distributions: (1) the distribution of distances between image features  $D \in \mathcal{D}_j$  and their centroid  $\bar{D}_j$  (red), and (2) the distribution of distances between centroids  $\bar{D}_j$  and the nearest centroid of a different point (blue). Although the expected distance (red) from a given descriptor to its centroid

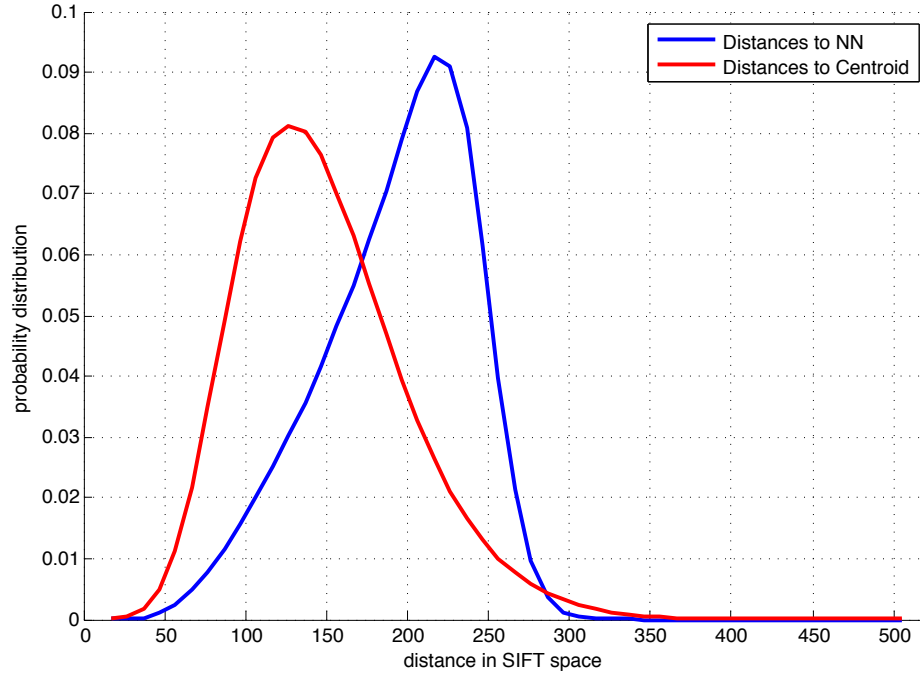


Figure 4.1: **Distributions of descriptor distances.** The curve in red shows the distribution of descriptor distances between image feature descriptors and their associated point descriptor centroids. The curve in blue shows the distribution of distances between a point descriptor centroid and its nearest neighbor in the full database. Note the significant overlap between these distributions. These distributions are generated from the Dubrovnik dataset by considering all database points and their associated image feature descriptors. (Figure best viewed in color.)

(true match) is smaller than the expected distance (blue) between two nearby centroids (false match), there is significant overlap between these two distributions. While this simple analysis is not a comprehensive study of feature mismatches, it suggests that there is significant opportunity for query features to match to incorrect points (i.e., because a feature is closer to a nearby, incorrect, database point in descriptor space).

One way to increase the probability of query features matching to the correct database point is to select points that are far away from each other in descrip-

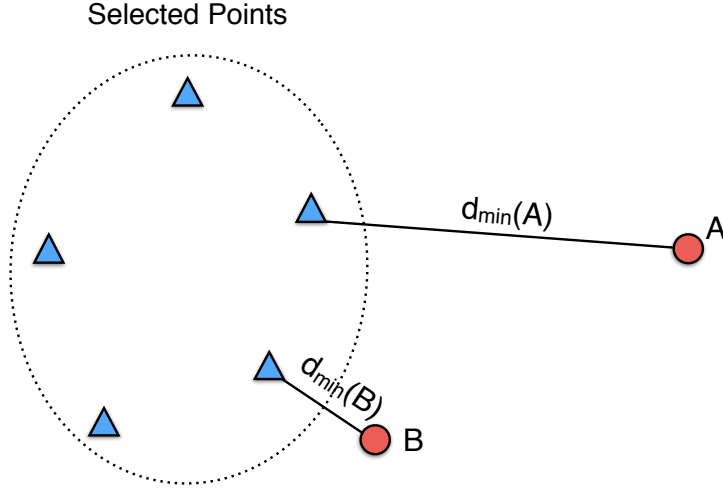


Figure 4.2: **An illustration of appearance-aware point selection.** The image above shows points in feature descriptor space (reduced to 2D for visualization purposes). Blue triangles represent point descriptors that are already selected by our algorithm, while red circles represent descriptors of candidate points to select next. Suppose that candidate points A and B cover exactly the same number of images in  $\mathcal{I} \setminus C$ , and thus would lead to equal gains in the  $K$ -cover algorithm. However, since the minimum distance of A to the selected point set  $d_{\min}(A)$  is larger than that of B  $d_{\min}(B)$ , A is likely to result in fewer mismatches during feature matching if selected. Hence A is preferred by our appearance-aware point selection algorithm.

tor space. Since our greedy selection algorithm adds points to  $\mathcal{P}'$  sequentially, when computing the gain of a point  $P_j$  under consideration, we implement this strategy by down-weighting a point's gain according to its minimum distance to the current set of selected points  $\mathcal{P}'$ . Figure 4.2 illustrates this intuition.

We evaluated a range of options for this weighting approach. In the end, we found that a simple approach worked well: let  $d_{\min}(j)$  be the minimum distance in descriptor space between  $P_j$  and the already selected point set  $\mathcal{P}'$ . We down-weight the gain of  $P_j$  if  $d_{\min}(j)$  is lower than a threshold  $d$ . Specifically, we define



the gain of a point  $P_j$  as:

$$G_{KCD}(j, \mathcal{P}') = w_d(d_{\min}(j))G_{KC}(j, \mathcal{P}') \quad (4.4)$$

where  $G_{KC}(j, \mathcal{P}')$  is defined in Eq. 4.1, and the weight  $w_d(d_{\min}(j))$  is defined as

$$w_d(d_{\min}(j)) = \begin{cases} d_{\min}(j)/d, & d_{\min}(j) < d \\ 1, & d_{\min}(j) \geq d \end{cases} \quad (4.5)$$

This weight varies from 0 to 1 linearly in the range  $[0, d]$ . One simple interpretation of this weight is as a rough approximation of the chance of a correct feature match for query features—higher for points that are more distinct given the current set of descriptors, and lower for points that are less distinct—and thus the gain function above can be interpreted as an “expected gain” in image coverage, incorporating the possibility of a mismatch. Given this interpretation, the threshold  $d$  should be set so as to try and separate the two distance distributions in Figure 4.1. We choose  $d = 180$  based on the empirical overlap of the two distributions.

We use this modified gain function in our greedy  $K$ -cover algorithm (the gain function  $G_{KCD}$  in Eq. 4.4 stands for “ $K$ -cover with distinctiveness”). There is an order dependency in our greedy algorithm, but since the order in which points are added depends largely on their coverage, our approach can be seen as a trade-off between our two main objectives of coverage and distinctiveness.

Like the  $K$ -cover algorithm, our modified covering algorithm terminates if the gain for *every* point is zero (i.e., no unchosen points will cover any not-yet-fully-covered images). Figure 4.3 shows the effect of including descriptor distances into the selection method, by showing distributions of distances between nearest neighbors in selected point sets with and without considering distinctiveness. We see that the point set selected by our method has larger expected

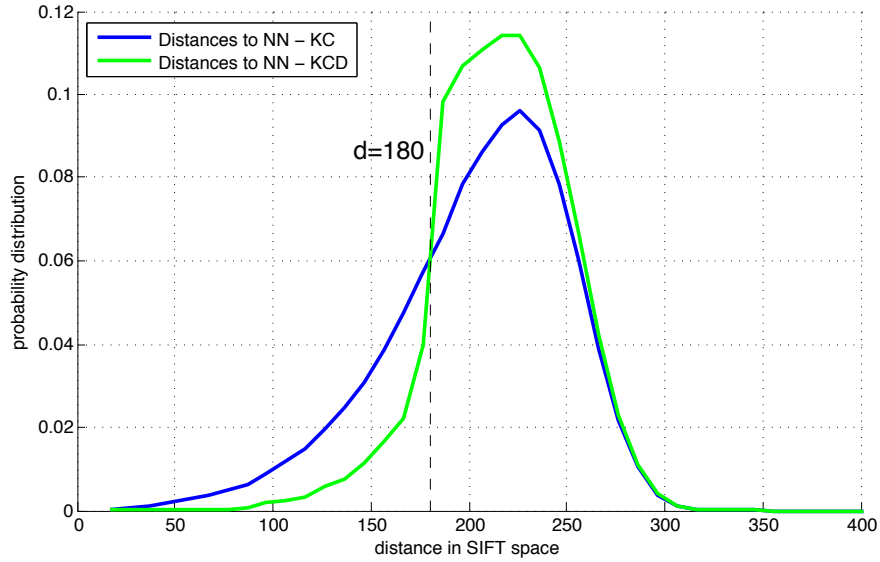


Figure 4.3: **Distributions of distances with and without appearance-aware selection.** This plot illustrates the “density” of two sets of point descriptors, by showing the distribution of distances between nearest neighbors in SIFT space between points within each set. The blue plot is for the set selected with the basic  $K$ -cover algorithm (KC), while the green plot is for the set selected with our appearance-aware selection algorithm (KCD) using the threshold  $d = 180$ . Both point sets contain 311,343 points selected from the Landmarks dataset [30]. Note that the KCD algorithm pushes points further away from one another on average.

nearest neighbor distances in descriptor space, which will tend to decrease the rate of false matches in the feature matching phase of the recognition pipeline. We use this appearance-aware selection method to seed our probabilistic point selection algorithm, which we describe next.

### 4.2.3 Probabilistic $K$ -cover algorithm

We now have an initial point set selected by our appearance-aware selection algorithm (KCD). This allows us to bootstrap our probabilistic point selection

method. Recall that rather than treating the visibility matrix as binary, our probabilistic approach treats this matrix as a set of noisy observations of visibility, and selects a small number of additional points to add to  $\mathcal{P}'$  such that the number of images that satisfy  $\Pr(v_{i,\mathcal{P}'} \geq K) \geq p_{\min}$  is as large as possible. That is, unlike the  $K$ -cover algorithm, which seeks to combinatorially “cover” the images at least  $K$  times, we set a minimum probability value  $p_{\min}$  and our goal is to achieve  $\Pr(v_{i,\mathcal{P}'} \geq K) \geq p_{\min}$  for each image  $I_i$ .

Like the  $K$ -cover algorithm, we use a greedy approach, but choosing the point  $P_{j^*}$  that maximizes *expected gain*, as defined in Eq. 4.3. This gain function is defined in terms of probabilistic coverage,  $\Pr(v_{i,\mathcal{P}'} \geq K)$ , which, in its simplest form, is a sum over a binomial distribution for  $x \geq K$ . In particular, given the initial set of points  $\mathcal{P}'$ , we first evaluate  $\Pr(v_{i,\mathcal{P}'} \geq K) = 1 - \Pr(v_{i,\mathcal{P}'} < K)$  for each image  $I_i$ , i.e. the probability that an image  $I_i$  sees at least  $K$  points in the selected point set  $\mathcal{P}'$ . Suppose  $I_i$  is covered  $C_i$  times by the initial point set  $\mathcal{P}'$ , and every edge encodes a point visibility with probability  $p$ . Then from the binomial distribution we have

$$\Pr(v_{i,\mathcal{P}'} = K') = \binom{C_i}{K'} p^{K'} (1-p)^{C_i-K'}. \quad (4.6)$$

Hence, we can compute  $\Pr(v_{i,\mathcal{P}'} \geq K)$  as

$$\Pr(v_{i,\mathcal{P}'} \geq K) = \sum_{K'=K}^{C_i} \Pr(v_{i,\mathcal{P}'} = K'). \quad (4.7)$$

These distributions for images with different levels of coverage are illustrated in Figure 4.4.

To choose the next point to add to  $\mathcal{P}'$ , we pick the point  $P_{j^*}$  that maximizes the sum of expected gains for all images, defined in Eq. 4.3. To compute this expected gain inside our greedy algorithm, the naive approach is to re-calculate

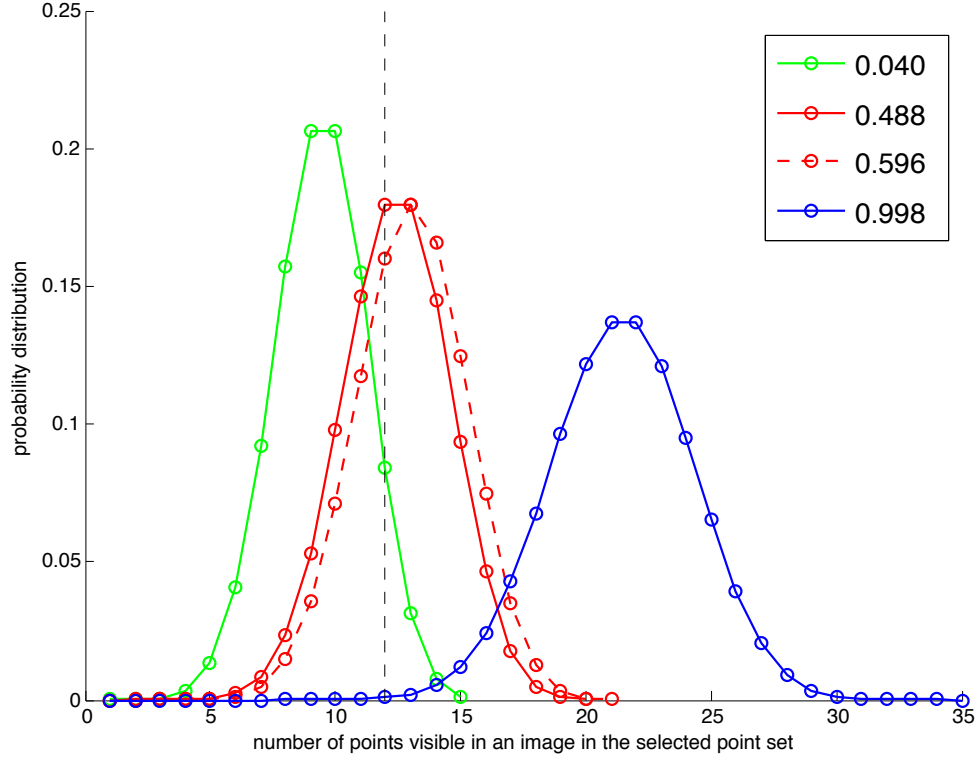


Figure 4.4: **Binomial distributions for images covered by different numbers of selected points.** The plots above show probability distributions for number of visible points,  $v_{i,\mathcal{P}'}$ , for three different images (solid lines). The images corresponding to the green, red solid, and blue curves are covered by 15, 20, and 35 points, respectively; in this example the probability of a positive point observation is set to  $p = 0.6$ . The legend shows the probability mass on the right side of the line  $K = 12$  for each image, i.e.  $\Pr(v_{i,\mathcal{P}'} \geq K)$ , the probability that each image sees at least  $K$  points in  $\mathcal{P}'$ . As more points are added, the distributions will shift from left to right (green  $\rightarrow$  red  $\rightarrow$  blue). The dotted red distribution is the red distribution after adding a single new point that is visible in this image; the corresponding probability value  $\Pr(v_{i,\mathcal{P}'} \geq K)$  has increased from 0.488 to 0.596 (a gain of 0.108). The objective of our probabilistic point selection algorithm is to select points that maximize the increase in expected gain  $\Pr(v_{i,\mathcal{P}'} \geq K)$  for all uncovered images.

(4.7) for all images. However, evaluating (4.7) can be expensive, as this must be done at each iteration of adding points, once for each point. In the simple case where  $p_{ij}$  is constant over the visibility graph (and hence the probability distributions of interest are binomial), we can simply pre-compute all possible distributions, each with a different  $C_i$ , and use these as lookup tables. However, for better generality and extensibility to Poisson binomial distributions, our implementation uses a different approach to compute  $\Pr(v_{i,\mathcal{P}'} \geq K)$ .

First, for each image  $I_i$ , we compute and store the full distribution of  $v_{i,\mathcal{P}'}$  once, after the initial points are selected, using (4.6). Then we re-use and update the distribution of  $v_{i,\mathcal{P}'}$  in later iterations. More specifically, from the independence assumption,  $\Pr(v_{i,\mathcal{P}' \cup \{j\}} \geq K)$  can be written as

$$\Pr(v_{i,\mathcal{P}' \cup \{j\}} \geq K) = p_{ij} \Pr(v_{i,\mathcal{P}'} = K - 1) + \Pr(v_{i,\mathcal{P}'} \geq K). \quad (4.8)$$

Hence, the gain of a point  $P_j$  for an image  $I_i$  is simply  $p_{ij} \Pr(v_{i,\mathcal{P}'} = K - 1)$ . Hence from Eq. 4.3, the total gain  $G_{KCP}(j, \mathcal{P}')$  of a point  $P_j$  can be written as

$$G_{KCP}(j, \mathcal{P}') = \sum_{i \in \mathcal{I} \setminus C} p_{ij} \Pr(v_{i,\mathcal{P}'} = K - 1). \quad (4.9)$$

We then choose the point  $P_{j^*}$  that maximizes  $G_{KCP}(j, \mathcal{P}')$ , add  $P_{j^*}$  to the set  $\mathcal{P}'$ , and update the distribution of  $v_{i,\mathcal{P}' \cup \{P_{j^*}\}}$  for each image  $I_i$  that  $P_{j^*}$  is visible in, using

$$\begin{aligned} \Pr(v_{I_i, \mathcal{P}' \cup \{P_{j^*}\}} = K') = \\ p_{ij^*} \Pr(v_{i,\mathcal{P}'} = K' - 1) + (1 - p_{ij^*}) \Pr(v_{i,\mathcal{P}'} = K'). \end{aligned} \quad (4.10)$$

**Discussion.** We have considered several variants of our approach. For instance, it would be natural to add the appearance distinctiveness weighting into the

probabilistic  $K$ -cover approach, and to that end, we have tried methods such as converting the minimum distance to the nearest neighbor to a probability using a global distribution of such distances. We found that our simple approach above worked as well, however, perhaps because the additional points we add are primarily improving coverage of a set of points that are already distinctive. However, this is an interesting topic for further exploration.

#### 4.2.4 Full point set reduction algorithm

0. Initialize point set  $\mathcal{P}' = \emptyset$ .

**Initial point set selection:**

1. Given  $K$  and threshold  $d$ , select the point  $P_{j^*} \in \mathcal{P}$  that maximizes  $G_{KCD}(j, \mathcal{P}')$  (Eq. 4.4), and add  $P_{j^*}$  to  $\mathcal{P}'$ .
2. Repeat Step 1 until all images are covered by at least  $K$  points.

**Probabilistic  $K$ -cover algorithm:**

3. Given a parameter  $p_{\min}$ , and the point set  $\mathcal{P}'$  generated from Steps 1 and 2, evaluate  $\Pr(v_{i,\mathcal{P}'} \geq K)$  for each image  $I_i$  using (4.7), and mark those images with  $\Pr(v_{i,S} \geq K) \geq p_{\min}$  as covered.
4. Select the point  $P_{j^*}$  that maximizes the gain function  $G_{KCP}(j, \mathcal{P}')$  defined in (4.9), and add  $P_{j^*}$  to  $\mathcal{P}'$ .
5. For each image  $I_i$  that sees point  $P_{j^*}$ : update  $I_i$ 's distribution using (4.10), re-evaluate  $\Pr(v_{i,\mathcal{P}'} \geq K)$  using (4.7) and mark  $I_i$  as covered if  $\Pr(v_{i,\mathcal{P}'} \geq K) \geq p_{\min}$ .
6. Repeat from Step 4 until a specified percentage of images are covered.

### 4.3 Implementation

**Efficient descriptor comparisons.** Our initial point set selection method requires computing the descriptor distance between a candidate point and its nearest neighbor for each point in the selected set, and a naive approach would involve comparing all candidate descriptors to all selected descriptors.<sup>1</sup> However, we note that the expected gain  $G_{KCD}(j, \mathcal{P})$  of a point  $P_j$  can only decrease across the iterations of the selection process, because both terms in Eq. 4.4 are submodular set functions of  $\mathcal{P}$ , which will only increase in its size. Hence, our algorithm can maintain an upper bound on the expected gain  $G_{KCD}(j, \mathcal{P})$  for each point  $P_j$ . At each iteration of searching for the best point to add, we can skip considering a point if its upper bound is less than or equal to the gain function value of the current best candidate point. The upper bound for each point is initialized by that point’s degree, the maximum possible score for each point. This bound is updated on any selection iteration where that point is not skipped. Using this method, a large number of points can be skipped; we observe empirically that approximately  $\mathcal{O}(\log n)$  points are evaluated per iteration, where  $n$  is the total number of points in the input set  $\mathcal{P}$ .

To further speed up our method, for each point  $P_j$  we also store the nearest neighbor and its corresponding distance in the selected set  $\mathcal{P}$  so far, as well as the size of  $\mathcal{P}$  during the last evaluation of  $P_j$ . This allows us to pick up where we left off when finding  $P_j$ ’s nearest neighbor the next time we evaluate  $P_j$ . All in all, we found the running time of our appearance-aware initial point selection process to be acceptable. For example, for the Dubrovnik dataset with  $K < 20$ , the selection process runs in under a minute. For  $K = 80$ , the process takes

---

<sup>1</sup>One could use a  $kd$ -tree to speed up nearest neighbor computation, but in our case the tree would have to be dynamic since the selected point set grows over time.

about ten minutes. An upper bound on running time of our KCD algorithm is  $\mathcal{O}(nc)$ , where  $n$  is the number of points in  $\mathcal{P}$  and  $c$  is the number of selected points ( $|\mathcal{P}'|$ ), since each selected point is compared to at most  $n$  other points.

**Parameters.** In all experiments, we define  $p_{\min} = 0.99$  to be the minimum probability for an image to be “covered”. We use 99% as the target coverage termination condition: that is, the  $K$ -cover algorithm terminates when 99% images are covered at least  $K$  times, and our algorithm terminates when 99% images are covered at least  $K$  times with probability greater than  $p_{\min}$ . We use a constant value  $p = 0.6$  for all  $p_{ij}$ 's<sup>2</sup>, and a descriptor distance threshold  $d = 180$  for Eq. 4.5. We evaluated several values of  $d$ , and found that the results are fairly insensitive to its value.

---

<sup>2</sup>We arrived at  $p=0.6$  by considering the empirical ratio between the number of inlier points when registering a query image and the number of points seen by that image in the original model; the results across a few datasets were in the range 0.5-0.6, and 0.6 worked well in practice.



Dataset	# DB Imgs	# 3D Points	# Queries
Dubrovnik [29]	6,044	1,886,884	800
Aachen [46]	4,479	1,980,036	369
Landmarks [30]	205,813	38,190,865	10,000

Table 4.1: Summary of datasets used in our experiments.

## 4.4 Experiments

In this section, we evaluate the performance of our algorithm on several datasets, including the Dubrovnik dataset of Li et al. [29], the Aachen dataset of Sattler et al. [46], and the much larger Landmarks dataset [30]; these three datasets are summarized in Table 4.1.

We evaluate three approaches to computing minimal scene descriptions: the  $K$ -cover algorithm (KC) [29], our initial point set selection algorithm only (KCD), and our full approach including the probabilistic  $K$ -cover algorithm (KCP). All methods output a list of points to keep in the original 3D point cloud database. We use each subset of points to construct a reduced database, and use the algorithm of [30] to register the query images for each dataset. We record the percentage of successfully registered images and use it as a measure of how well the point set represents the original database. We are particularly interested in *very compact* scene descriptions (small  $K$ ), and understanding how well we can represent scenes with a small fraction of points.

**Numbers of points.** In order to fairly compare different methods, it is easiest to compare the performance of scene descriptions with the same number of points. However, given a particular  $K$ , the number of points required to cover a database is generally smaller for the  $K$ -cover algorithm (KC) than with KCD, since KC selects points with maximal coverage without considering point

appearance. Hence to compare performance, we run KCD until it selects the same number of points as the KC algorithm with the same  $K$  value. This could slightly favor the  $K$ -cover algorithm, as our initial point set selection algorithm is terminating early.

Since our full approach (KCP) consists of two stages, in which the initial selection KCD alone selects slightly more points than the KC algorithm, again, more points will be selected by KCP compared to KC using the same  $K$  value. To account for this, we use a lower value of  $K$  to select the initial point set (around  $pK$ ), and continue running the KCP algorithm until it has selected the same number of points as KC. We show results on all datasets in Table 4.4.

For each dataset, we plot starting with the smallest  $K$  where we get close to 50% registration rate. Hence the  $K$  values vary for different datasets. Table 4.4 shows the results for KC, KCD, and KCP.

**Initial point set selection.** In all datasets, adding the descriptor distance-based weight  $w(d_{\min}(j))$  in our gain function (4.4) for KCD improves the performance compared to the  $K$ -cover algorithm for nearly all values of  $K$ . The improvement is especially significant when  $K$  is low (and hence the number of selected points is small). However, as  $K$  is set higher, this advantage becomes less prominent, perhaps because images see more points and mismatches are less detrimental (i.e., coverage starts to win out).

**Probabilistic  $K$ -cover.** Table 4.4 also shows that our full approach (KCP) consistently outperforms the  $K$ -cover algorithm, and further improves on the gains achieved by our KCD algorithm. For instance, KCD improves recognition performance on the Dubrovnik dataset by nearly 5% (58% to 62.9%) when  $K = 12$ ,

Dubrovnik Dataset [29]				
# query images: 800, registered by full set: 99.50%				
$K$	12 (9)	20 (12)	30 (20)	50 (35)
# points	5,788	10,349	17,241	31,752
% points	0.31%	0.55%	0.91%	1.68%
KC	58.00%	77.06%	86.00%	91.81%
KCD	62.88%	78.88%	<b>87.38%</b>	92.50%
KCP	<b>64.25%</b>	<b>79.13%</b>	87.25%	<b>93.38%</b>

Aachen Dataset [46]				
# query images: 369, registered by full set: 88.08%				
$K$	30 (20)	50 (32)	80 (52)	100 (65)
# points	13,299	23,675	40,377	52,161
% points	0.67%	1.20%	2.04%	2.63%
KC	50.95%	62.06%	66.40%	71.27%
KCD	54.20%	63.14%	69.38%	72.36%
KCP	<b>56.37%</b>	<b>64.23%</b>	<b>70.19%</b>	<b>73.98%</b>

Landmarks Dataset [30]				
# query images: 10,000, registered by full set: 94.33%				
$K$	6 (4)	9 (6)	12 (9)	20 (12)
# points	140,306	222,161	311,035	571,864
% points	0.37%	0.58%	0.81%	1.50%
KC	44.84%	59.86%	69.56%	81.06%
KCD	45.45%	61.26%	70.59%	81.04%
KCP	<b>45.90%</b>	<b>61.50%</b>	<b>71.87%</b>	<b>81.45%</b>

Table 4.2: **Registration performance on Dubrovnik, Aachen, and Landmarks datasets.** **KC** stands for the  $K$ -cover algorithm, **KCD** stands for our appearance-aware point selection algorithm, and **KCP** stands for our full approach. Point sets of the same size are selected using the three algorithms, then used in the same registration algorithm [30] to evaluate the percentages of query images that are successfully registered to the database. Smaller  $K$  values (in brackets) are used to initialize our KCP method. For each experiment, we show the number of points in the reduced model, the percentage of total points this represents, and the performance of the three methods. For comparison, we also show the performance of [30] using the full set of input points.

and KCP improves performance further to 64.2%. To check whether KCD is indeed helping, we tried initializing KCP with KC instead of KCD, but found that this performs worse than KCP initialized with KCD.

How much performance are we losing with our compactness? Table 4.4 also shows the performance of state-of-the-art methods that utilize the full point set [30, 45], which use full models and have a minimum registration rate of 88% on these datasets. It is worth noting that although compared to them, our method has lower raw registration performance (Table 4.4), our resulting models are much more compact ( $< 3\%$  of the size of the full model). With the limited portion of database we use, our algorithm still performs surprisingly well in registering new images. For instance, we can recognize over 70% of the query images in the Aachen dataset with only 2% of the 3D points in the full model ( $K = 80$ ).

How much benefits are we getting through compactness? As well as dramatically improving memory use, we have also observed reduced registration time. For instance, the smaller reduced models ( $K \leq 30$ ) process queries in about half the time compared to the full model. This improvements result from the compactness of data structures and efficiency in rejecting false images, both of which stem from the compactness of the 3D point set.

## CHAPTER 5

### CONCLUSION

In this thesis, we have shown that graph is a useful abstraction for modeling image collections using machine learning methods. Even with small amounts of training data, discriminative learning techniques are particularly useful in both improving the efficiency of producing image graphs and learning good representations of the neighborhoods in the graph.

Reasoning using graph also provides an alternative way of thinking in formulating computer vision tasks such as classification or categorization, where each category is treated mostly independently. As has been shown in Chapter 3, graphs are capable of encoding rich structural information, which can be summarized or learned using learning techniques in a natural way. This work also points to several future research directions, such as better definition of graphs for different tasks, better learning methods to learn useful models from these graphs, or better image representation in which space the learning will occur.

For large scale recognition problems, scalability is a key factor in determining whether a system is applicable or not. It is a very interesting problem of maintaining most information while reducing the size of the database as much as possible. In Chapter 4, we showed that, again, reasoning about graphs is useful. The main difference in this setting is that we reason about image-point graph, instead of image graph, which is used in previous two chapters.

In conclusion, the world is inter-connected, and the visual world is especially so. For this reason, graphs are interesting tools for the researchers to model and understand images. There are also many interesting challenges and opportu-

nities in achieving this goal. For example, graph structured image data can be naturally combined with unsupervised learning to discover interesting features of visual world, such as categories, clusters, trends etc. Supervised learning can also benefit from graph structure, which will provide the learning algorithm a rich form of supervision, and possibly combined with user's feedback to perform online supervised learning to, in turn, consistently update and improve the graph's structure. Scalability is still a practical issue to deal with no matter what the system want to achieve in today's era of big data. Therefore, how to best summarize and describe the image data using graphs is also a very interesting direction to explore.

## APPENDIX A

### A.1 Derivation for the iterative usage of the update factor

Here we provide the full derivation of how our diversity reranking method updates the probability scores for each image each time we select a new image for the shortlist. Let  $n$  denote the number of images that have already been selected for the ranked shortlist  $RL = \{i_1, i_2, \dots, i_n\}$ . For any image  $u$  in the as-yet-unselected set of images  $\mathcal{I} \setminus RL$ , we wish to compute

$$P_u^{(n)} = \Pr(X_u = 1 | X_{i_1} = 0, \dots, X_{i_n} = 0),$$

which is the conditional probability of image  $u$  matching the query image given that all the  $n$  images in the current ranking list do **not** match the query. To simplify the analysis, we separately consider the cases  $n = 1$ ,  $n = 2$ , and the general case  $n = k$  (note that in Section 3.4 we denote  $P_u^{(1)}$  as  $P'_u$ ).

- $n = 1$ : From Eq. (3.2), Section 3.4 derives the update factor in Eq. (3.3) used to compute  $P_i^{(1)}$ :

$$P_u^{(1)} = P_u \frac{1 - \frac{P_{u,i_1}}{P_u} P_{i_1}}{1 - P_{i_1}}$$

where  $P_u$  is the original probability estimate from our GBP approach.

- $n = 2$ : From the definition of conditional probability, we have that

$$\begin{aligned} P_u^{(2)} &= \Pr(X_u = 1 | X_{i_2} = 0, X_{i_1} = 0) \\ &= \frac{\Pr(X_u = 1, X_{i_2} = 0, X_{i_1} = 0)}{\Pr(X_{i_2} = 0, X_{i_1} = 0)} \\ &= \frac{\Pr(X_u = 1, X_{i_2} = 0 | X_{i_1} = 0) \Pr(X_{i_1} = 0)}{\Pr(X_{i_2} = 0, X_{i_1} = 0)} \end{aligned} \tag{A.1}$$

Recall from Section 3.4 that we make two simplifying assumptions:

1. *Independence of  $(X_{i_1} = 0)$  and  $(X_{i_2} = 0)$  for distant  $i_1$  and  $i_2$ :*

$$\Pr(X_{i_2} = 0, X_{i_1} = 0) = \Pr(X_{i_2} = 0) \Pr(X_{i_1} = 0)$$

2. *Independence of  $(X_u = 1, X_{i_2} = 1)$  and  $(X_{i_1} = 0)$ :*

$$\Pr(X_u = 1, X_{i_2} = 1 | X_{i_1} = 0) = \Pr(X_u = 1, X_{i_2} = 1)$$

Given these assumptions, we can simplify Eq. (A.1) as follows:

$$\begin{aligned}
P_u^{(2)} &= \Pr(X_u = 1 | X_{i_2} = 0, X_{i_1} = 0) \\
&= \frac{\Pr(X_u = 1, X_{i_2} = 0 | X_{i_1} = 0) \Pr(X_{i_1} = 0)}{\Pr(X_{i_2} = 0) \Pr(X_{i_1} = 0)} \\
&= \frac{\Pr(X_u = 1, X_{i_2} = 0 | X_{i_1} = 0)}{\Pr(X_{i_2} = 0)} \\
&= \frac{\Pr(X_u = 1 | X_{i_1} = 0) - \Pr(X_u = 1, X_{i_2} = 1 | X_{i_1} = 0)}{1 - \Pr(X_{i_2} = 1)} \\
&= \frac{\Pr(X_u = 1 | X_{i_1} = 0) - \Pr(X_u = 1, X_{i_2} = 1)}{1 - \Pr(X_{i_2} = 1)} \\
&= \frac{P_u^{(1)} - \Pr(X_u = 1 | X_{i_2} = 1) \Pr(X_{i_2} = 1)}{1 - \Pr(X_{i_2} = 1)} \\
&= \frac{P_u^{(1)} - P_{u,i_2} P_{i_2}}{1 - P_{i_2}} = P_u^{(1)} \left( \frac{1 - \frac{P_{u,i_2}}{P_u^{(1)}} P_{i_2}}{1 - P_{i_2}} \right) \tag{A.2}
\end{aligned}$$

where  $P_u^{(1)}$  is the updated conditional probability of  $X_u = 1$  using the evidence  $X_{i_1} = 0$ . Note that the update factor at the end of Eq. (A.2) is of the same form of Eq. (3.3).

- $n = k$ : More generally, we can similarly derive that

$$P_u^{(k)} = P_u^{(k-1)} \left( \frac{1 - \frac{P_{u,i_k}}{P_u^{(k-1)}} P_{i_k}}{1 - P_{i_k}} \right) \tag{A.3}$$

Hence, through Eq. (A.3), we can update each as-yet-unchosen image  $u$  conditional probability  $P_u^{(k)}$  in a iterative fashion each time we add a new image to the ranking list  $RL$ .



## BIBLIOGRAPHY

- [1] S. Agarwal, N. Snavely, I. Simon, S.M. Seitz, and R. Szeliski. Building Rome in a day. In *ICCV*, 2009.
- [2] R. Arandjelovic and A. Zisserman. Three things everyone should know to improve object retrieval. In *CVPR*, 2012.
- [3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. of the ACM*, 45(6):891–923, 1998.
- [4] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Supervised semantic indexing. In *CIKM*, 2009.
- [5] S. Cao and N. Snavely. Learning to match images in large-scale collections. In *ECCV Workshop on Web-scale Vision and Social Media*, 2012.
- [6] Song Cao and Noah Snavely. Graph-based discriminative learning for location recognition. In *CVPR*, 2013.
- [7] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. An online algorithm for large scale image similarity learning. In *NIPS*, 2009.
- [8] Harr Chen and David R Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *ACM SIGIR*, 2006.
- [9] Siddharth Choudhary and PJ Narayanan. Visibility probability structure from SfM datasets and applications. In *ECCV*. 2012.
- [10] O. Chum and J. Matas. Unsupervised discovery of co-occurrence in sparse high dimensional data. In *CVPR*, 2010.
- [11] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, pages 1–8, 2007.
- [12] Ondrej Chum and Jirí Matas. Large-scale discovery of spatially related images. *PAMI*, 2010.
- [13] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What makes Paris look like Paris? *SIGGRAPH*, 2012.

- [14] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 2008.
- [15] J.-M. Frahm et al. Building Rome on a cloudless day. In *ECCV*, 2010.
- [16] A. Frome and J. Malik. Learning distance functions for exemplar-based object recognition. In *ICCV*, 2007.
- [17] A. Frome, Y. Singer, F. Sha, and J. Malik. Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *ICCV*, 2007.
- [18] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 1998.
- [19] M. Havlena, A. Torii, and T. Pajdla. Efficient structure from motion by graph optimization. In *ECCV*, 2010.
- [20] J. Hays and A.A. Efros. Im2gps: estimating geographic information from a single image. In *CVPR*, 2008.
- [21] A. Irschara, C. Zach, J.M. Frahm, and H. Bischof. From structure-from-motion point clouds to fast location recognition. In *CVPR*, 2009.
- [22] H. Jegou, M. Douze, and C. Schmid. On the burstiness of visual elements. In *CVPR*, 2009.
- [23] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Improving bag-of-features for large scale image search. *IJCV*, 87(3):316–336, feb 2010.
- [24] E. Johns and G.Z. Yang. From images to scenes: Compressing an image cluster into a single scene model for place recognition. In *ICCV*, 2011.
- [25] J. Knopp, J. Sivic, and T. Pajdla. Avoiding confusing features in place recognition. In *ECCV*, 2010.
- [26] Fayin Li and Jana Kosecka. Probabilistic location recognition using reduced feature set. In *ICRA*, 2006.
- [27] Xiaowei Li, Changchang Wu, Christopher Zach, Svetlana Lazebnik, and

- Jan-Michael Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV*, 2008.
- [28] Y. Li, D.J. Crandall, and D.P. Huttenlocher. Landmark classification in large-scale image collections. In *ICCV*, 2009.
  - [29] Y. Li, N. Snavely, and D. Huttenlocher. Location recognition using prioritized feature matching. In *ECCV*, 2010.
  - [30] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua. Worldwide pose estimation using 3d point clouds. In *ECCV*, 2012.
  - [31] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.
  - [32] T. Malisiewicz and A. Efros. Beyond categories: The visual memex model for reasoning about object relationships. *NIPS*, 2009.
  - [33] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-SVMs for object detection and beyond. In *ICCV*, 2011.
  - [34] A. Mikulik, M. Perdoch, O. Chum, and J. Matas. Learning a fine vocabulary. In *ECCV*, 2010.
  - [35] N. Naikal, A. Yang, and S.S. Sastry. Informative feature selection for object recognition via sparse PCA. In *ICCV*, 2011.
  - [36] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.
  - [37] Hyun Soo Park, Yu Wang, Eriko Nurvitadhi, James C Hoe, Yaser Sheikh, and Mei Chen. 3d point cloud reduction using mixed-integer quadratic programming. In *CVPR Workshops*, 2013.
  - [38] J. Philbin, O. Chum, M. Isard, J.ivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007.
  - [39] J. Philbin, O. Chum, M. Isard, J.ivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008.

- [40] J. Philbin, M. Isard, J. Sivic, and A. Zisserman. Descriptor learning for efficient retrieval. In *ECCV*, 2010.
- [41] J. Philbin, J. Sivic, and A. Zisserman. Geometric latent dirichlet allocation on a matching graph for large-scale image datasets. *IJCV*, 2010.
- [42] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 1999.
- [43] S. E. Robertson. Readings in information retrieval. chapter The Probability Ranking Principle in IR, pages 281–286. 1997.
- [44] T. Sattler, B. Leibe, and L. Kobbelt. Fast image-based localization using direct 2D-to-3D matching. In *ICCV*, 2011.
- [45] T. Sattler, B. Leibe, and L. Kobbelt. Improving image-based localization by active correspondence search. In *ECCV*, 2012.
- [46] T. Sattler, T. Weyand, B. Leibe, and L. Kobbelt. Image retrieval for image-based localization revisited. In *BMVC*, 2012.
- [47] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR*, 2007.
- [48] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2003.
- [49] Abhinav Shrivastava, Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Data-driven visual similarity for cross-domain image matching. *SIGGRAPH ASIA*, 2011.
- [50] I. Simon, N. Snavely, and S.M. Seitz. Scene summarization for online image collections. In *ICCV*, 2007.
- [51] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.
- [52] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *SIGGRAPH*, 2006.

- [53] A. Torii, J. Sivic, and T. Pajdla. Visual localization by linear combination of image descriptors. In *ICCV Workshops*, 2011.
- [54] P. Turcot and D.G. Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems. In *Workshop on Emergent Issues in Large Amounts of Visual Data, ICCV*, 2009.
- [55] Michel Vidal-Naquet and Shimon Ullman. Object recognition with informative features and linear classification. In *ICCV*, 2003.
- [56] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. *NIPS*, 2003.
- [57] J. Zhang, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *IJCV*, 73, 2007.
- [58] W. Zhang and J. Kosecka. Image based localization in urban environments. In *Int. Symp. on 3DPVT*, 2006.
- [59] Yan-Tao Zheng et al. Tour the world: building a web-scale landmark recognition engine. In *CVPR*, 2009.